

IBM RT Advanced Interactive Executive Operating System Version 2.2

AIX Operating System Technical Reference

Files and Extensions

Programming Family

IBM

SC23-2126-0

IBM RT Advanced Interactive Executive Operating System Version 2.2

AIX Operating System

Technical Reference

Files and Extensions

Programming Family

IBM

Second Edition (September 1988)

Portions of the code and documentation described in this book were developed at the Electrical Engineering and Computer Sciences Department at the Berkeley Campus of the University of California under the auspices of the Regents of the University of California.

The Rand MH Message Handling System was developed by the Rand Corporation and the University of California.

The Network File System was developed by Sun Microsystems, Inc.

This edition applies to Version 2.2 of the IBM AIX Operating System. Changes are made periodically to the information herein; these changes will be reported in technical newsletters or in new editions of this publication.

References in this publication to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM licensed program in this publication is not intended to state or imply that only IBM's licensed program may be used. Any functionally equivalent program may be used instead.

International Business Machines Corporation provides this manual "as is," without warranty of any kind, either express or implied, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this manual at any time.

Products are not stocked at the address given below. Requests for copies of this product and for technical information about the system should be made to your IBM authorized RT dealer, your IBM marketing representative, or your IBM authorized remarketer.

A reader's comment form is provided at the back of this publication. If the form has been removed, address comments to IBM Corporation, Department 997, 11400 Burnet Road, Austin, Texas 78758-3493. IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you.

© Copyright International Business Machines Corporation 1985, 1987, 1988

© Copyright INTERACTIVE Systems Corporation 1984

© Copyright AT&T Technologies 1984

Trademarks

The following trademarks apply to this book:

- AIX is a trademark of International Business Machines Corporation.
- DEC and VT100 are trademarks of Digital Equipment Corporation.
- Hayes is a trademark of Hayes Microcomputer Products, Inc.
- IBM is a registered trademark of International Business Machines Corporation.
- NFS is a trademark of Sun Microsystems, Inc.
- PC-XT is a trademark of International Business Machines Corporation.
- Personal Computer AT is a registered trademark of International Business Machines Corporation
- Proprinter is a trademark of International Business Machines Corporation.
- Racal-Vadic is a trademark of Racal-Milgo.
- RT is a registered trademark of International Business Machines Corporation.
- Smartmodem is a trademark of Hayes Microcomputer Products, Inc.
- Sun is a trademark of Sun Microsystems, Inc.
- Tektronics is a trademark of Tektronix, Inc.
- UNIX was developed and licensed by AT&T. It is a registered trademark of AT&T in the United States of America and other countries.

About The AIX Operating System

The AIX Operating System is based on UNIX System V with many enhancements. Figure 1-1 shows the primary components of the AIX Operating System and their relationships.

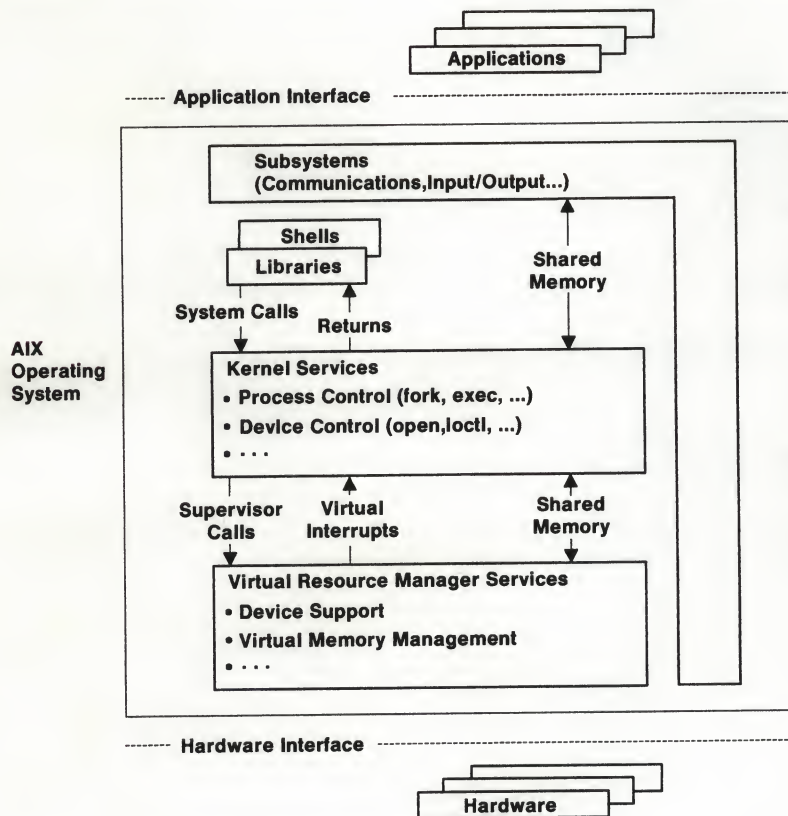


Figure 1-1. AIX Operating System Structure

The AIX Operating System is composed of the kernel, the Virtual Resource Manager (VRM), shells, libraries, and various subsystems. The combined function and features of these interdependent components form the AIX Operating System.

The functional characteristics and interface relationships of the AIX Operating System components are the product of an implementation designed to maximize:

- Source code portability
- Efficiency of overall system operation and use.

AIX directs and controls all RT hardware and software through functions and services assigned to the kernel and VRM components. Although one of these components may provide more functionality than the other component for a particular service, both components are typically required to provide the service.

The kernel provides many services, including:

- Process control
- Device control
- File system management
- Memory management
- Time management
- Program management
- Resource management.

The kernel supports the VRM component in several ways. The kernel maintains files used by the VRM, provides dynamic configuration services, and provides protection and security mechanisms.

The VRM supports the kernel by providing basic and enhanced services, including:

- Device support
- Virtual memory management
- Real-time support
- Multiple kernel support.

The documentation for the IBM RT AIX Operating System is contained in the four-volume *AIX Operating System Technical Reference*. These volumes are:

- *System Calls and Subroutines*
- *Files and Extensions*
- *VRM Programming Support*
- *VRM Device Support*.

About This Book

Audience and Purpose

This book provides information on application programming interfaces to the Advanced Interactive Executive Operating System (referred to in this text as AIX) for use on the RT.

This book is intended for experienced C programmers. To use this book effectively, you should be familiar with AIX or UNIX System V commands, system calls, subroutines, file formats, and special files. If you are not already familiar with AIX or UNIX System V, see *Using the AIX Operating System*.

How To Use This Book

The contents of Chapters 2, 3, 4, 5, and 6 are organized in alphabetical order by name. Related information in those chapters is combined into one description where applicable, but each item appears as a separate entry in the Index.

Organization

This book is divided into two volumes, *System Calls and Subroutines*, and *Files and Extensions*. Each volume contains a table of contents, list of figures, and an index for both volumes.

Volume 1 contains Chapters 1 and 2; volume 2 contains Chapters 3 through 6 and Appendixes A through E. The subjects of these chapters are:

- Chapter 1, "AIX Operating System," gives an overview of the operation and organization of the kernel.
- Chapter 2, "System Calls and Subroutines," describes the C language interface to the operating system calls, subroutines, and macros available to C language programmers.
- Chapter 3, "File Formats," defines the formats of various system and user files.
- Chapter 4, "Miscellaneous Facilities," includes miscellaneous information, such as text-processing macro packages.
- Chapter 5, "Special Files," describes special files associated with specific peripheral devices and device drivers.
- Chapter 6, "Advanced Display Graphics Support Library," describes the Advanced Display Graphics Support Library.

In the chapters that are organized alphabetically, some entries describe several facilities (system calls, subroutines, file formats, or special files). In such cases, each facility appears only once, alphabetized under its *major* name. If you have difficulty finding a given facility, look it up in the index at the back of the book.

All entries are based on a common format, but all of the sections do not always appear.

Purpose

Briefly describes the purpose of the facility.

Syntax or Synopsis

Shows how to use the facility. See “Syntax” on page ix for details about the information given in the “Syntax” sections for system calls and subroutines.

Description

Describes the system call, subroutine, file format, or special file in detail.

Return Value

Explains the value returned by a system call or subroutine.

Diagnostics

Lists the possible values that a system call can return in the **errno** external variable if an error occurs. See Appendix A, “Error Codes,” for a complete list of these values.

Examples

Shows one or more examples of how to use the facility.

Files

Lists the names of files that the facility uses.

Related Information

Refers you to additional information you may find helpful. This section refers you first to additional topics in this book, then to other publications. References from this book are given in the order they appear in the book, alphabetically within each chapter.

Volume 2 also contains the following appendixes:

- Appendix A, “Error Codes,” lists and describes the values that system calls pass back when they encounter error conditions.
- Appendix B, “Writing a Queuing System Backend,” provides detailed information about writing friendly backend programs.
- Appendix C, “Porting DOS 3.0 Applications,” explains how to port programs from the IBM Personal Computer Disk Operating System (DOS) to the AIX Operating System.
- Appendix D, “Component Cross-Reference,” lists the programs with which certain subroutines and subroutine libraries are packaged.
- Appendix E, “Glossary,” defines terms to help you better understand AIX and the RT work station.

A Reader's Comment Form and Book Evaluation Form are provided at the back of each volume of this book. Use the Reader's Comment Form at any time to give IBM information that may improve the book. After you have become familiar with the book, use the Book Evaluation Form to give IBM specific feedback about the book.

Syntax

The "Syntax" section of each system call and subroutine entry in this book gives the syntax needed to invoke it. The following conventions are used in this section:

Boldface type shows text to be entered exactly as shown.

Italic type shows parameters that should be replaced with actual values.

[] (square brackets) enclose optional parameters.

. . . (an ellipsis) follows a parameter that can be repeated any number of times.

The information shown in each "Syntax" section is usually the set of declarations as they might appear in the actual C language definition of the call or subroutine. These declarations give you more information than showing the exact calling sequence as it appears in a user program.

Consider the following example of a subroutine entry:

```
#include <stdio.h>
```

```
FILE *fopen (path, type)  
char *path, *type;
```

The **#include** statement names a header file that contains definitions needed by the subroutine. See "Header Files" on page x for more information.

The first line following the **#include** statement shows the data type of the return value (**FILE ***), the name of the subroutine (**fopen**), and the parameters that it takes (*path* and *type*). The following lines indicate the data type of each parameter. The fact that the name **FILE** is in all capitals indicates that this data type is defined in the **stdio.h** header file.

This subroutine might actually be used in a program like this:

```
#include <stdio.h>

. . .
static char filename[ ] = "test.data";
main ( )
{
    FILE *inputfile;

    . . .
    inputfile = fopen (filename, "r+");
    . . .
}
```

Note that the type of both parameters is stated as **char *** (pointer to character), but that the value given for each is actually a pointer to a character string (an array of characters). In the C language, pointers and arrays are treated similarly so that the notations ***p** and **p[0]** are generally interchangeable. Thus, when this book shows a parameter of type **char ***, a character string is frequently required. Check the "Description" section to make sure.

Because **fopen** returns a type other than **int**, the subroutine must be declared so that the compiler knows this information. In this particular case, it is already declared for you in the header file. Sometimes, however, you may need to declare a system call or subroutine yourself. For this example, the declaration would take the form:

```
FILE *fopen();
```

Such a declaration should be put at the top of your program, before any references to the system call or subroutine. Note that the declaration resembles the syntax shown in this book, except that the parameters are omitted and a semicolon is added to the end.

See *C Language Guide and Reference* or another C language manual for more detailed information about pointers, arrays, and subroutine declarations.

Header Files

Many system calls and subroutines require that header files be included in the programs that use them. When this is the case, the **#include** statements needed are shown in the "Syntax" section of the call or subroutine entry. Consider the following example:

```
#include <stdio.h>
```

When a program is being compiled, this **#include** statement inserts the text of the **stdio.h** header file into the source program. The **< >** delimiters indicate that the file is located in the **/usr/include** directory. All of the header files used by the system calls and

subroutines described in this book are located in **/usr/include** or in one of its subdirectories.

The header files contain definitions of constants and macros that the C language preprocessor interprets. The **#include** statements must precede all references in your program to the constants and macros that the header files define. Most of the time you can simply put all of the **#include** statements together at the top of the program. If you use several calls or subroutines that require the same header files, then each file should be included only once. If a system call or subroutine requires more than one header file, be careful to enter the **#include** statements in the order shown.

By convention, the names of most of the constants are in capital letters. Therefore, a name that appears in this book in bold capitals (for example, **EFAULT**) is a constant defined in a header file. A few constants are not named in capitals, notably **stdin**, **stdout**, and **stderr**, which are defined in **stdio.h**.

The constant **NULL** is commonly used to denote a null pointer value. This book sometimes mentions **NULL** when discussing system calls and subroutines that do not require header files. If you compile a program and get an error message indicating that **NULL** is not defined, insert the following statement before the first **NULL** in your program:

```
#define NULL 0
```

In addition to constants, header files sometimes define macros and data types. Macros take parameters and resemble subroutines, but there are several differences:

- You cannot take the address of a macro with the C language **&** operator.
- The parameters of a macro are evaluated in a different manner from those of a subroutine. See *C Language Guide and Reference* or another C language manual for details about parameter evaluation.

This book describes a macro as a subroutine, then adds the statement that it is implemented as a macro.

Related Publications

Related information can be found in the following books:

- *IBM RT Using the AIX Operating System* describes using the AIX Operating System commands, working with file systems, developing shell procedures, and using data communications facilities.
- *IBM RT Managing the AIX Operating System* provides instructions for performing such system management tasks as adding and deleting user IDs, creating and mounting file systems, repairing file system damage, and managing data communications facilities.

-
- *IBM RT AIX Operating System Programming Tools and Interfaces* describes the programming environment of the AIX Operating System and includes information about using the operating system tools to develop, compile, and debug programs. In addition, this book describes the operating system services and how to take advantage of them in a program. This book also includes a diskette that includes programming examples, written in C language, to illustrate using system calls and subroutines in short, working programs. (Available optionally)
 - *IBM RT AIX Operating System Commands Reference* lists and describes the AIX Operating System commands.
 - *IBM RT C Language Guide and Reference* provides guide information for writing, compiling, and running C language programs and includes reference information about C language data structures, operators, expressions, and statements. (Available optionally)
 - *IBM RT Assembler Language Reference* describes the IBM RT Assembler Language and includes descriptions of syntax and semantics, machine instructions, and pseudo-operations. This book also shows how to link and run Assembler Language programs, including linking to programs written in C language. (Available optionally)
 - *IBM RT Keyboard Description and Character Reference* describes the national character and keyboard support for the 101-key, 102-key, and 106-key keyboards, including keyboard position codes, keyboard states, control code points, code sequence processing, and nonspacing character sequences. (Available optionally)
 - *IBM RT Hardware Technical Reference* is a three-volume set. Volume I describes how the system unit operates, including I/O interfaces, serial ports, memory interfaces, and CPU interface instructions. Volumes II and III describe adapter interfaces for optional devices and communications and include information about IBM Personal Computer family options and the adapters supported by 6151 and 6150. (Available optionally)
 - *IBM RT Messages Reference* lists messages displayed by the IBM RT and explains how to respond to the messages.
 - *IBM RT Installing and Customizing the AIX Operating System* provides step-by-step instructions for installing and customizing the AIX Operating System, including how to add or delete devices from the system and how to define device characteristics. This book also explains how to create, delete, or change AIX and non-AIX minidisks.
 - *IBM RT Personal Computer AT Coprocessor Services Technical Reference* describes differences between writing applications for the IBM Personal Computer AT and IBM RT Personal Computer AT Coprocessor Services. This book also includes Personal Computer AT Coprocessor Services BIOS listing, hardware technical information, Math Co-Processor information, and the 286 and 287 instruction sets. (Available optionally)
 - *IBM RT Using DOS Services* provides step-by-step information for using AIX Operating System DOS Services. (Available optionally; packaged with *IBM RT DOS Services Reference*)

-
- *IBM RT DOS Services Reference* provides reference information about the AIX Operating System DOS Services. This book also includes information on sharing DOS files with Personal Computer AT Coprocessor Services, and on the differences between PC DOS and DOS Services. (Available optionally; packaged with *IBM RT Using DOS Services*)
 - *IBM RT Interface Program for use with TCP/IP* describes the Interface Program commands for transferring data among host computers, logging into remote computers, executing commands remotely, and managing networks. This book also describes the programming interfaces to the Interface Program. (Available optionally)
 - *IBM RT Device Driver Development Guide* provides information about incorporating support for additional input/output devices into an AIX environment on the IBM RT. Topics covered include device drivers (kernel device heads and VRM device handlers), device managers, LPOSTs, display modules, configuration, installation, debugging, and display support for GSL. This book includes diskettes containing sample C-language programs that illustrate device drivers and GSL display support. (Available optionally)

See *IBM RT Bibliography and Master Index* for order numbers of RT publications and related diskettes.

Ordering Additional Copies of This Book

To order additional copies of this publication, use either of the following sources:

- To order from your IBM representative, use Order Number SBOF-1823.
- To order from your IBM authorized RT work station dealer, use Part Number 27F4355.

The binders and *AIX Operating System Technical Reference* manuals are included with the order. For information on ordering the binders or manuals separately, contact your IBM representative or your IBM dealer.

Contents

Volume 1. System Calls and Subroutines

Chapter 1. AIX Operating System	1-1
Chapter 2. System Calls and Subroutines	2-1
Index	X-1

Volume 2. Files and Extensions

Chapter 3. File Formats	3-1
Chapter 4. Miscellaneous Facilities	4-1
Chapter 5. Special Files	5-1
Chapter 6. Advanced Display Graphics Support Library	6-1
Appendix A. Error Codes	A-1
Appendix B. Writing a Queuing System Backend	B-1
Appendix C. Porting DOS 3.0 Applications	C-1
Appendix D. Component Cross-Reference	D-1
Appendix E. Glossary	E-1
Index	X-1

Figures

1-1.	AIX Operating System Structure	v
2-1.	Floating-Point Trap Handler Structures	2-316
2-2.	The fpfp Register Mapping	2-318
2-3.	Default Error-Handling Procedures	2-440
2-4.	How to Assign Program Numbers	2-618
2-5.	Structure of the Ring Error Monitor	2-662
2-6.	Ring Error Monitor Calling Sequence	2-664
2-7.	REM Timer Subsystem Timer Processing	2-673
2-8.	API Control Subroutine Calls to Set REM Parameters	2-675
2-9.	API Control Subroutine Calls to Set API Parameters	2-675
2-10.	REM Output Buffer Structure Definition	2-677
3-1.	Example of Annotated Text Font Storage (non-5081)	3-84
3-2.	Example of Annotated Text Font Storage (5081)	3-86
3-3.	Example of an rtfont Pel Box	3-99
3-4.	Information Record Format	3-117
4-1.	Octal ASCII Character Set	4-3
4-2.	Hexadecimal ASCII Character Set	4-4
4-3.	Code Page P0	4-6
4-4.	Code Page P1	4-7
4-5.	Code Page P2	4-8
4-6.	Code Page P0	4-27
4-7.	Code Page P1	4-34
4-8.	Code Page P2	4-41
4-9.	EBCDIC Character Set	4-46
4-10.	The eqnchar Characters	4-56
4-11.	Greek Characters	4-62
5-1.	Bit Positions of ASCII Controls in Echo Map	5-52
5-2.	Screen Manager Ring Examples	5-70
5-3.	Position Codes for Remapping a Keyboard	5-97
6-1.	Default Attribute Values	6-11
D-1.	Extended Services Subroutines	D-1
D-2.	Multi-User Services Subroutines	D-1

Volume 2. Files and Extensions

Chapter 3. File Formats

About This Chapter

This chapter outlines the formats of various files. The C language **struct** declarations for the file formats are given where applicable. These structures are usually found in header files located in the **/usr/include** or **/usr/include/sys** directories, although they can be located in any directory in the file system.

Many of the files described in this chapter contain ***magic numbers*** at predefined offsets. Magic numbers provide programs with a way to verify the format of an input file before attempting to process it. The values used for magic numbers are chosen because they are not likely to occur as a random pattern in normal input.

.init.state

Purpose

Specifies the initial state for the AIX Operating System.

Description

The `/etc/.init.state` file specifies the initial state in which the `init` process is to start up the AIX Operating System. This file contains a single line that specifies one of the following initial states:

- m** Maintenance mode (single-user mode). The `/etc/rc` shell procedure is not run when you enter maintenance mode. Therefore, no devices are configured, no file systems are mounted, and no daemon processes are started. See the `/etc/rc` file on your system for the commands that perform these operations.
- a** Automatic multiuser mode. Enters multiuser mode, passing an `a` to `/etc/rc` as the first parameter, `$1`.
- c** "Clean" multiuser mode. Enters multiuser mode, passing a `c` to `/etc/rc` as the first parameter, `$1`. By convention, the `c` indicates that the file systems are probably in good condition, or "clean." You can customize the `/etc/rc` file on your system to skip running `fsck` in order to shorten the system startup procedure. Note, however, that "clean" mode does *not* guarantee that any file systems are in good condition.
- d** "Dirty" multiuser mode. Enters multiuser mode, passing a `d` to `/etc/rc` as the first parameter, `$1`. By convention, the `d` indicates that one or more file systems may have been damaged. `/etc/rc` should run the `fsck` command to check all file systems.
- e file** Exec mode. Executes the shell procedure named *file*. When the shell procedure terminates, the system asks the operator whether to enter maintenance mode or multiuser mode.
- o** Operator mode. Asks the operator whether to enter maintenance mode or multiuser mode. The system waits until a response is given.
- u** Unknown mode. Asks the operator whether to enter maintenance mode or multiuser mode. If no response is given within a period of time (approximately a minute), the system enters automatic multiuser mode.

.init.state

If the operator selects multiuser mode in response to a prompt, then **init** asks whether to check the file systems. The response to this question determines whether a C or a d is passed to **/etc/rc**.

The **/etc/.init.state** file can also contain comment lines, which are indicated by a # (pound sign) in the first column.

File

/etc/.init.state

Related Information

In this book: “Creation and Execution” on page 1-12, “iplvm, waitvm” on page 2-409, and “reboot” on page 2-599.

The **init** and **rc** commands in *AIX Operating System Commands Reference*.

a.out

Purpose

Provides common assembler and link editor output.

Synopsis

```
#include <a.out.h>
```

Description

The **as** (assembler) and **ld** (link editor) programs produce an output file (the **a.out** file by default) in the following format. The **a.out** file is executable if the assembler and the link editor do not find any unresolved external references or errors in the source.

This file can have the following sections: a header, the text segment, data segment, relocation information, a symbol table, a line number section, a string table, and a shared library identifier (in that order). The last five sections may be missing if the program was linked with the **-s** flag of the **ld** command or if they were removed by the **strip** command. The shared library identifier exists only for object modules related to a shared library image. Note the relocation information is not present if there are not external references to be resolved after linking.

Loading an **a.out** file into memory for execution causes the creation of three logical segments: the text segment, the data segment (initialized data followed by data that is not initialized, the latter actually being initialized to all zeros) and a stack.

Segment 1 occupies a low memory address in the process image; its size is static. Segment 2 follows segment 1 in memory. The size of this segment can be extended using the **brk** system call. The stack segment begins near the highest locations in segment 3 and grows toward segment 2 as required.

Header

The format of the **a.out** header is:

```
struct exec {
    unsigned char a_magic[2]; /* magic number */
    unsigned char a_flags;    /* flags */
    unsigned char a_cpu;      /* CPU-ID */
    unsigned char a_hdrlen;   /* length of header */
    unsigned char a_unused;   /* reserved for future use */
};
```

a.out

```
    unsigned short a_version; /* version stamp */
    long a_text;           /* size of text segment */
    long a_data;           /* size of data segment */
    long a_bss;            /* size of bss segment */
    long a_entry;          /* entry point */
    long a_misc;           /* misc., e.g. initial stack pointer */
    long a_syms;           /* symbol table size */
    /* SHORT FORM ENDS HERE */
    long a_trsize;         /* text relocation size */
    long a_drsize;         /* data relocation size */
    long a_tbase;          /* text relocation base */
    long a_dbase;          /* data relocation base */
    long a_lnums;          /* size of line number section */
    long a_toffs;          /* offset of text from start of file */
};
```

The fields in the header are as follows:

- a_magic** A 2-byte number that has a value of 0x0103.
- a_flags** A byte with various options that apply to the **a.out** file. Bits that are not used are set to 0. Options supported are:
- A_TOFF** Text offset is specified by **a_toffs**
 - A_STRS** String table is present
 - A_HDREXT** Extended header is present
 - A_EXEC** File is executable
 - A_SEP** Instruction and data spaces are separate
 - A_PURE** Pure text
 - A_SHLIB** Shared library identifier is present
- a_cpu** A coded entry describing the system unit and the byte order it expects. The coded entry for the RT system is 0x13.
- a_hdrlen** The length of the header. The size of the header is variable, but it must be at least 32 bytes to include all of the fields in the structure through **a_syms**. If the size of the header is such that a field is not included, the default value is assumed.
- a_misc** The maximum size in bytes the user stack is allowed to grow.

Extended Header

An extended header is indicated by the A-HDREXT bit being set in **a-flags**. The format of the extended header is:

```
struct exthdr {
    unsigned short ax-size;      /* total size of extension */
    unsigned short ax-type;      /* type of extension */
    unsigned short ax-flags;     /* e.g., execution model */
    unsigned short ax-nsecs;     /* number of segment entries */
};
```

The size of the extension (in bytes) is **ax-size**, which includes the length of **exthdr** plus any auxiliary entries which comprise this extended header type, indicated by **ax-type**. The value of **ax-flags** is also dependent on **ax-type**. In the event that the following auxiliary entries contain per-segment information, **ax-nsecs** is the number of segments (and thus the number of auxiliary entries) present.

Legal values for **ax-type** are:

```
AXT-INTEL    1
AXT-SHLIB    2
```

The legal values for **ax-flags** when **ax-type** is AXT-INTEL are:

```
AXF-SSS      Separate stack segment
AXF-MCS      Multiple code segments
AXF-MDS      Multiple data segments
AXF-HDS      Huge data present
AXF-OVLY     Code overlay
AXF-FPH      Floating-point hardware required
AXF-ABS      Absolute addresses present
```

When **ax-type** is AXT-INTEL, **exthdr** is followed by **ax-nsecs** entries of the form:

```
struct segent {
    unsigned short as-type;      /* segment type */
    unsigned short as-flags;     /* segment attributes */
    unsigned short as-num;       /* segment number */
    unsigned short as-nlnno;     /* # lineno entries */
    long as-filep;               /* position (offset) in file */
    long as-psize;               /* size of segment in file */
    long as-vsize;               /* virtual size */
    long as-rsvd1;               /* reserved */
    long as-rsvd2;               /* reserved */
    long as-lnptr;               /* position of lineno entries */
};
```

```
};
```

Each **segment** describes a segment of the **a.out** file. Legal values for the type of segment, **as_type**, are:

```
AST_NULL
AST_TEXT    Code segment
AST_DATA    Data segment
```

Various characteristics of the segment are described by **as_flags**. Possible values are:

```
ASF_HUGE      Segment contains huge model data
ASF_BSS       Segment contains implicit bss
ASF_SHARE     Segment is sharable
ASF_EXPDOWN   Segment expands downward
ASF_SEG       Always on for segments
```

When **ax_type** is **AXT-SHLIB**, **exthdr** is followed by a table describing the **ax-nsegs** shared libraries required by this program. Each element of the table has the format:

```
struct slent {
    long sl_off;      /* offset from table start of lib key */
    long as_addr;     /* address where library to be mapped */
};
```

The table is terminated by an element with an **sl_off** member of zero. Following the table are the shared library keys associated with the libraries mentioned. Each shared library key is preceded by a string recognizable to the **what** command, and is terminated with an ASCII NUL character. (Each **sl_off** entry points past the **what** string to the real start of the key.)

Text and Data Sections

The text and data sections are indicated in the fields as follows:

a_text	The size of the text segment in bytes. This segment begins immediately after the header or at the offset specified in the a_toffs field if the A_TOFF flag is set. The A_TEXTPOS macro defined in the a.out.h header file gives the offset of this segment in either case.
a_data	The size of the data segment in bytes. This segment begins immediately following the text segment. The A_DATAPOS macro gives the offset of this segment.
a_bss	The size of the bss segment in bytes. This segment represents data that is not initialized. It does not appear in the file.

The text, data, and bss segments must each be several full words in length.

- a_entry** The text address where the program should start to run. The default is the *a_tbase* value.
- a_tbase** The virtual address of the first byte of the text segment. The default value for this field is 0.
- a_dbase** The virtual address of the first byte of the data segment. The default value for this field is **a_tbase** + **a_text**, rounded to the next segment boundary.

Relocation

The fields in the relocation information are as follows:

- a_drsize** The size of the data relocation information in bytes. The **A_DRELPOS** macro defines where the data relocation information entries begin.
- a_trsize** The size of the text relocation information in bytes. The **A_TRELPOS** macro defines where the text relocation entries begin.

A word in the text or data segment of memory contains either an actual value or the value of an offset. If a word in the text or data segment references an undefined external symbol, its value is an offset from the associated external symbol. During processing, the link editor defines the external symbol and adds the value of the symbol to the word in the file.

When relocation information is present, each item that can be relocated is 8 bytes long. The format of the relocation information is:

```
struct reloc {
    long r_vaddr;           /* virtual address of reference */
    unsigned short r_symndx; /* internal segnum or extern
                             symbol number */
    unsigned short r_type;  /* relocation type */
};
```

The **r_vaddr** field gives the location of the relocatable reference relative to the beginning of the segment in which it is defined.

The **r_symndx** field contains a symbol number in the case of an external. Otherwise, it contains a segment number code:

S_ABS	0xFFFF	/* absolute */
S_TEXT	0xFFFE	/* text segment */
S_DATA	0xFFFD	/* data segment */
S_BSS	0xFFFC	/* bss segment */

The **r_type** field indicates the type of relocation. The relocation types are:

R_ABS	0	/* absolute */
R_RELBYTE	2	/* byte */
R_PCRBYTE	3	/* byte (pc relative) */
R_RELWORD	4	/* word */
R_PCRWORD	5	/* word (pc relative) */
R_RELLONG	6	/* long */
R_PCRLONG	7	/* long (pc relative) */
R_REL3BYTE	8	/* 3 bytes */
R_KBRANCH	9	/* 20-bit 1-shifted */
R_SEG86	10	/* segmented PC-XT */
R_SEG286	11	/* segmented Personal Computer AT */
R_KCALL	12	/* 20-bit 1-shifted or fix up */

Symbol Table

The **a_syms** field in the header indicates the size of the symbol table in bytes. The **A_SYMPOS** macro defines the offset where the symbol table begins.

The symbol table consists of the following entries:

```
struct syment {
    union {
        char _n_name [8];      /* non-flex version */
        struct {
            long _n_zeroes;    /* flexname == 0 */
            long _n_offset;    /* offset into string table */
        } _n_n;
        char *_n_n_ptr[2];     /* allows for overlaying */
    } _n;
    long n_value;              /* symbol value */
    unsigned char n_sclass;    /* storage class */
    unsigned char n_numaux;    /* number of auxiliary entries */
    unsigned short n_type;     /* language base and derived type */
};

#define SYMENT struct syment
#define SYMESZ sizeof(struct syment)
```

```
#define n_name _n._n-name
#define n_nptr _n._n-nptr[1]
#define n_zeroes _n._n-n._n-zeroes
#define n_offset _n-n._n-n._n-n-offset
```

The low-order 3 bits of **n-*sclass*** indicate the section information:

N_UNDF	00	/* undefined */
N_ABS	01	/* absolute */
N_TEXT	02	/* text */
N_DATA	03	/* data */
N_BSS	04	/* bss */
N_COMM	05	/* common */
N_SECT	07	/* section mask */

The high-order bits indicate the storage class. The following storage classes are implemented:

C_NULL	0000	/* undefined symbol */
C_AUTO	0010	/* (0x08) automatic variable */
C_EXT	0020	/* (0x010) external symbol */
C_STAT	0030	/* (0x18) static */
C_REG	0040	/* (0x20) register variable */
C_EXTDEF	0050	/* (0x28) external definition */
C_LABEL	0060	/* (0x30) label */
C_ULABEL	0070	/* (0x38) undefined label */
C_MOS	0100	/* (0x40) member of structure */
C_ARG	0110	/* (0x48) function argument */
C_STRTAG	0120	/* (0x50) structure tag */
C_MOU	0130	/* (0x58) member of union */
C_UNTAG	0140	/* (0x60) union tag */
C_TPDEF	0150	/* (0x68) type definition */
C_USTATIC	0160	/* (0x70) undefined static */
C_ENTAG	0170	/* (0x78) enumeration tag */
C_MOE	0200	/* (0x80) member of enumeration */
C_REGPARM	0210	/* (0x88) register parameter */
C_FIELD	0220	/* (0x90) bit field */
C_BLOCK	0300	/* (0xc0) .bb or .eb */
C_FCN	0310	/* (0xc8) .bf or .ef */
C_EOS	0320	/* (0xd0) end of structure */
C_FILE	0330	/* (0xd8) file name */
N_CLASS	0370	/* (0xff) storage class mask */

If a symbol section and class is **undefined external** and the value field is a value other than 0, the link editor interprets the symbol as the name of a common region in which the size is indicated by the value of the symbol.

The **n-type** field is primarily for use by a symbol debugger. The low-order 4 bits form the base type with values defined as follows:

T_NULL	0	/* undefined symbol */
T_ARG	1	/* used internally by compiler */
T_CHAR	2	/* character */
T_SHORT	3	/* short integer */
T_INT	4	/* integer */
T_LONG	5	/* long integer */
T_FLOAT	6	/* floating point */
T_DOUBLE	7	/* double */
T_STRUCT	8	/* structure */
T_UNION	9	/* union */
T_ENUM	10	/* enumeration */
T_MOE	11	/* member of enumeration */
T_UCHAR	12	/* unsigned character */
T_USHORT	13	/* unsigned short */
T_UINT	14	/* unsigned integer */
T_ULONG	15	/* unsigned long */

The high-order bits form the derived type. The following values are repeated up to six times to form the derived type:

DT_NON	0	/* no derived type */
DT_PTR	1	/* pointer */
DT_FCN	2	/* function */
DT_ARY	3	/* array */

The **n-numaux** field contains the number of auxiliary entries associated with this symbol table entry. Currently, a symbol table entry can have at most one auxiliary entry. The auxiliary entry provides additional information, and has this form:

```
union auxent {
    struct {
        long x_tagndx;           /* str, union, or enum tag index */
        union {
            struct {
                ushort x_lnno;   /* declaration line number */
                ushort x_size;   /* str, union, array size */
            } x_lnsz;
            long x_fsize;        /* size of function */
        } x_misc;
    };
};
```

```

union {
    struct {
        long x_lnnoptr;      /* if ISFCN, tag, or .bb */
        long x_endndx;      /* ptr to fcn line # */
        } x_fcn;            /* entry index past block end */
    struct {
        ushort x_dimen[DIMNUM];
        } x_ary;
    } x_fcenary;
} x_sym;
struct {
    char x_fname[FILNMLEN];
} x_file;
};

#define FILNMLEN    14
#define DIMNUM      4

```

The information in an auxiliary entry cannot be correctly interpreted without the symbol table entry to which it belongs. The order of entries within the symbol table is significant.

Line Number Section

The **a_lnums** field contains the size in bytes of the line number section. The line number section starts at the location in the file defined by the **A-LINEPOS** macro.

Line number entries are used by the symbolic debugger to debug code at the source level. Entries within the line number section are grouped by function. The format of a line number entry is:

```

struct lineno {
    union {
        long l_symndx;      /* symbol table index of function name
                             if and only if l_lnno == 0 */
        long l_paddr;      /* physical address of line number */
    } l_addr;
    unsigned short l_lnno;  /* line number */
};

```

String Table

The string table contains the names of symbols that are longer than 8 characters. It is present only if the **A_STRS** flag is set. If present, the first 4 bytes contain the length, in bytes, of the string table, including the count. The remainder of the table is a sequence of null-terminated strings. If the **n-zeroes** field in a symbol entry is 0, the **n-offset** field gives the offset into the string table of the name for the symbol.

Shared Library Identifier

The shared library identifier names the shared library image to which this object module is related. It is present only if the **A_SHLIB** flag is set. If present, the first byte contains the length of the identifier section including the count byte. The identifier itself is a string terminated with an ASCII NUL character.

Related Information

The **as**, **cc**, **config**, **dump**, **ld**, **nm**, **sdb**, **size**, **strip** and **what** commands in *AIX Operating System Commands Reference*.

acct

Purpose

Provides the accounting file format for each process.

Synopsis

```
#include <sys/acct.h>
```

Description

The accounting files provide a means to monitor the use of the system. These files also serve as a method for billing each process for processor usage, materials, and services. The **acct** system call produces accounting files. The `<sys/acct.h>` file defines the records in these files. The content of the records are:

```
/* Accounting structures */
typedef ushort comp_t;      /* floating point */
                             /* 13-bit fraction, 3-bit exponent */

struct acct
{
    char  ac_flag;          /* Accounting flag */
    char  ac_stat;          /* Exit status */
    ushort ac_uid;          /* Accounting user-ID */
    ushort ac_gid;          /* Accounting group-ID */
    dev_t ac_tty;           /* control typewriter */
    time_t ac_btime;        /* Beginning time */
    comp_t ac_utime;         /* accounting user time in clock ticks */
    comp_t ac_stime;         /* accounting system time in clock ticks */
    comp_t ac_etime;         /* accounting elapsed time in clock ticks */
    comp_t ac_mem;           /* memory usage */
    comp_t ac_io;            /* chars transferred */
    comp_t ac_rw;            /* blocks read or written */
    char  ac_comm[8];        /* command name */
};
```

```

extern struct acct acctbuf;
extern struct inode *acctp; /* i-node of accounting file */

#define AFORK 01           /* has executed fork, but no exec */
#define ASU 02            /* used superuser authority */
#define ACCTF 0300        /* record type: 00 = acct */

```

The fields are as follows:

- ac-comm** This field contains the command name. A child process, created by a **fork** system call, receives this information from the parent process. An **exec** system call resets this field.
- ac-flag** This field indicates whether the process used superuser authority, or it was created using a **fork** command but not yet followed by an **exec** system call. The **fork** command turns the **AFORK** flag in this field on and the **exec** system call turns the **AFORK** flag off.
- ac-mem** This field contains memory usage. For each clock tick, the system updates this field with the current process size and charges usage time to the process. This is computed as $((data\ size) + (text\ size)) \div (number\ of\ in-memory\ processes\ using\ text)$.

The following structure (not part of **acct.h**) represents the total accounting format used by the various accounting commands:

```

/* Float arrays below contain prime time and non-prime time
   components */

struct tacct {
    uid_t  ta-uid;           /* user-ID */
    char   ta-name[8];       /* login name */
    float  ta-cpu[2];        /* cum. CPU time, p/np (mins) */
    float  ta-kcore[2];      /* cum. kcore-mins, p/np */
    float  ta-io[2];         /* cum. chars xferred (512s) */
    float  ta-rw[2];         /* cum. blocks read/written */
    float  ta-con[2];        /* cum. connect time, p/np, mins */
    float  ta-du;            /* cum. disk usage */
    long   ta-qsys;          /* queuing sys charges (pgs) */
    float  ta-fee;           /* fee for special services */
    long   ta-pc;            /* count of processes */
    unsigned short ta-sc;    /* count of login sessions */
    unsigned short ta-dc;    /* count of disk samples */
};

```

File

`/usr/include/sys/acct.h`

Related Information

In this book: “acct” on page 2-22 and “utmp, wtmp, .ilog” on page 3-246.

The **acctcom** command in *AIX Operating System Commands Reference*.

The **acct**, **acctcms**, **acctcon**, **acctmerg**, **acctprc**, **acctsh**, **diskusg**, and **runacct** procedures in *AIX Operating System Commands Reference*.

ar

ar

Purpose

Describes common archive file format.

Synopsis

```
#include <ar.h>
```

Description

The **ar** (archive) command is used to combine several files into one. The **ar** command creates an **ar** file. The **ld** (link editor) searches archive files to resolve program linkage.

Each archive begins with the archive magic string:

```
#define ARMAG "!<arch>\n" /* magic string */
#define SARMAG 8 /* length of magic string */
```

Each archive that contains common object files includes an archive symbol table. See "a.out" on page 3-5 for the format of an object file. **ld** uses this symbol table to determine the archive members to load during the link edit process. The archive symbol table, if it exists, is always the first file in the archive. It is never listed, but **ar** automatically creates and updates it.

The archive file members follow the archive header and symbol table. A file member follows each file member header. The format of a file member header is:

```
#define ARFMAG "\n" /* header trailer string */

struct ar_hdr { /* file member header */
    char ar_name[16]; /* file member name - terminated by '/' */
    char ar_date[12]; /* file member date */
    char ar_uid[6]; /* file member user identification */
    char ar_gid[6]; /* file member group identification */
    char ar_mode[8]; /* file member mode */
    char ar_size[10]; /* file member size */
    char ar_fmags[2]; /* ARFMAG - string to end header */
};
```

All information in the file member header is in printable ASCII. The numeric information contained in the headers is stored as decimal numbers, except **ar_mode**, which is stored in octal. Thus, if the archive contains printable files, you can print the archive.

The **ar_name** field is blank-padded and terminated by a / (slash). The **ar_date** field indicates the date the file was last modified prior to archive. The **ar** command allows archives to move from system to system.

Each archive file member begins on an even-byte boundary. **ar** inserts null bytes for padding and a new-line character between files, if necessary. The **ar_size** field is the size of the file without padding. An archive file contains no empty areas.

If the archive symbol table exists, the first file in the archive has a zero-length name, for example, **ar_name[0] = '/'**. The contents of the symbol table are as follows:

- The number of symbols. This is 4 bytes long.
- The array of offsets into the archive file. The length is determined by 4 bytes times the number of symbols.
- The name string table. The size is determined by **ar_size** minus (4 bytes times (the number of symbols plus 1)).

The **sgetl** and **sputl** functions manage the number of symbols and the array of offsets. The string table contains an equal number of null-terminated strings and elements in the offsets array. Each offset from the array associates with the corresponding name from the string table, in order. The string table names all the defined global symbols found in the object files contained in the archive. Each offset locates the archive header for the associated symbol.

Related Information

In this book: “**sgetl**, **sputl**” on page 2-733 and “**a.out**” on page 3-5.

The **ar**, **ld**, and **strip** commands in *AIX Operating System Commands Reference*.

attributes

attributes

Purpose

Describes an attribute file format.

Description

ASCII files are used to control some RT utilities in order to simplify installing, customizing, and maintaining the RT system. A text editor can be used to examine or change these files. These files share an attribute-structured format.

An attribute-structured file consists of one or more named stanzas, separated by blank lines. Each stanza begins with a name followed by a colon, and contains assignments to keyword attributes. The values assigned can be alphanumeric strings or arbitrary character strings enclosed in double quotation marks. The assignments can associate either a single value or a succession of values with each attribute.

The size limits for stanzas are: a maximum of 400 keywords per stanza, a maximum of 4K bytes per stanza, and a maximum of 512 bytes per keyword.

Typically, the stanza name is the name of a device or service. The attributes describe the properties or handling of the named device or service. The meanings of the stanza names, attribute names, and values are specific to an application. Examples of this file type distributed with the system are */etc/filesystems*, */etc/ports*, and */etc/qconfig*.

The stanza name **default** can be used to specify default values for any attributes. These default assignments are implicitly included in every stanza that follows. A specified value overrides the default value. A new **default** stanza automatically cancels all previously specified default values. The syntax of a file of this type is:

```
<file>          ::= <stanza>
                ::= <file> <blank line> <stanza>
<stanza>        ::= <name>:
                ::= <name>:<assignments>
<name>          ::= file name or information similar in syntax
<assignments>  ::= <assignment>
                ::= <assignments> <assignment>
<assignment>   ::= <attribute>=<values>
<attribute>    ::= string of alphanumeric characters
```


<values>	::= <value>
	::= <values>,<value>
<value>	::= string of alphanumeric characters
	::= "arbitrary characters"

Lines beginning with an * (asterisk) are considered to be comments and are ignored.

Note: Make sure that the values assigned to attribute keywords do not contain blanks unless they are enclosed in double quotation marks.

Related Information

In this book: "cc.cfg" on page 3-31, "connect.con" on page 3-37, "filesystems" on page 3-74, "master" on page 3-125, "ports" on page 3-173, "qconfig" on page 3-185, "rasconf" on page 3-189, and "system" on page 3-210.

audit

audit

Purpose

Describes the format of audit bins and records.

Synopsis

```
#include <sys/audit.h >
```

Description

The file **/usr/include/sys/audit.h** contains descriptions of the format of the audit trail. The form of the bin's head and tail is described by `struct aud_bin`. The `bin_tail` field contains either a 0 (if the description is of the audit trail head) or a 1 (if the description is of the audit trail tail).

The form of the audit record is described by `aud_rec`.

Related Information

In this book: “audit” on page 2-31 and “auditlog” on page 2-39.

The **audit**, **auditpr**, and **pack** commands in the *AIX Operating System Commands Reference*.

autolog

Purpose

Performs login function automatically.

Description

The optional **autolog** file causes the RT system to perform a login sequence automatically when it contains a valid user name. When power is applied to the system and the login port is the console, **login** searches for this file. If this file is found, **login** creates a session for a specific user automatically. The **autolog** file is an ASCII file containing a valid user name. A user can create this file while customizing the system. After it is created, this file can be edited using any editor. If this file does not exist, **login** causes the user to login as usual.

File

`/etc/autolog`

Related Information

The **login** command in *AIX Operating System Commands Reference*.

backup

backup

Purpose

Copies file system onto temporary storage media.

Synopsis

```
#include <backup.h>
```

Description

A backup of the file system provides protection against substantial data loss due to accidents or error. The **backup** command writes file system backups and, conversely, the **restore** command reads file system backups. The following text describes the format of a file system backup.

Header Types

The backup contains several different types of header records along with the data in each file that is backed up. The types of header records are:

- | | |
|------------------|--|
| FS-VOLUME | The volume label. This header exists on every volume. |
| FS-FINDEX | An index of files on this volume. Multiple headers of this type can appear on a volume if there are too many i-nodes for the initial index. This header is followed by data. |
| FS-CLRI | A bit map of i-nodes on the file system. A zero bit indicates the i-node is not in use. This header exists only on the first volume. If the backup is a level-zero backup, this header is omitted. |
| FS-BITS | Another bit map of i-nodes. A one bit indicates the i-node is present on this volume or a subsequent volume. This header may not appear on all volumes. |
| FS-DS | Specifies the node ID of the node being backed up and the full path name of the directory that is being backed up. This header appears only on the first volume. |
| FS-VOLEND | Indicates the end of the current volume. This header may not appear on all volumes. This header is used to indicate that all index entries on this volume are used. |

FS-END	Indicates the end of the backup. This header appears on every volume.
FS-INODE	Describes a single i-node. This header is followed by data that consists of directories then followed by the other files within the directories.
FS-NAME	A description of a file that is backed up by name.

Header Sequence

The header sequence varies depending on whether the files are backed up by i-node or by name and on the type of backup device used.

Volume 1 of i-node backups to direct access volumes have the following sequence, assuming that more than one volume is required for backup:

FS-VOLUME
FS-CLRI
FS-BITS
FS-FINDEX, followed by data
FS-FINDEX (if applicable), followed by data
FS-END

Subsequent volumes have the following sequence:

FS-VOLUME, followed by data
FS-FINDEX, followed by data
FS-FINDEX (if applicable), followed by data
FS-END

I-node backups to tapes have the same format as previously described, except there are no **FS-FINDEX** headers and the **FS-BITS** header appears on every volume.

The format of backups by name does not depend on the output device. These backups have a simple format:

FS-VOLUME	Appears on each volume.
FS-DS	Appears on the first volume.
FS-NAME	Precedes the data for each file. The files are copied in the order they were named.
FS-END	Concludes the backup.

Header Format

The location and size of the headers are independent of any blocking for either the file system or the backup device. Each header begins on an 8-byte boundary. The length of a header depends on its type, but is always padded to a multiple of 8 bytes. Data from a file is similarly padded. Some headers contain addresses of other headers that are the offset in 8-byte units from the beginning of the backup volume.

Each field in a header is written in low-order bytes first for portability. I-node numbers within directories also follow this order. The header begins with the following structure:

```
struct hdr {
    unsigned char  len;
    unsigned char  type;
    ushort         magic;
    ushort         checksum;
};
```

The fields in this header indicate the following information:

len	The length of the header in 8-byte units.
type	The type of the header.
magic	The magic number, which identifies this file as a file system backup. The magic number is one of the following values: MAGIC Identifies this as a regular file system backup. PACKED-MAGIC Identifies this as a packed, or compressed, file system backup. Each data file within it is compressed using the same algorithm that is used by the pack command. Header information is not compressed.
checksum	A checksum.

Volume Headers

FS_VOLUME headers have the following structure:

```
struct {
    struct hdr h;
    ushort     volnum;
    time_t     date;
    time_t     budate;
    daddr_t     numwds;
    char        disk[16];
    char        fsname[16];
};
```



```

        char    user[16];
        short   incno;
    };

```

The fields contain the following information:

volnum Contains the volume number.

date Indicates the date the backup was made.

budate Indicates that all files changed since this date are backed up.

numwds Indicates the number of 8-byte words in this backup.

disk Identifies the device that was backed up.

fsname Identifies the logical name of the backed-up device, for example, /a.

user Identifies the user that made the backup.

incno Shows the level number of the backup.

For backups by name, **budate**, **disk**, and **fsname** have no meaning, and **incno** is 100.

Index Headers

FS_FINDEX records are as follows:

```

    struct {
        struct hdr h;
        ushort    dummy;
        ino_t      ino[80];
        daddr_t    addr[80];
        daddr_t    link;
    };

```

The fields are:

ino I-numbers of files indexed

addr Addresses of file indexed

link Address of next index on this volume, or 0 if this is the last.

Bit Maps

FS_CLRI and **FS_BITS** headers have the same structure:

```
struct {
    struct hdr h;
    ushort      nwds;
};
```

In both cases, the bit map follows the header, and **nwds** gives the length of the map in 8-byte units. To save space, some zero bits at the end of the map are not backed up.

Location Headers

FS_DS headers have the following format:

```
struct {
    struct hdr h;
    char      nid[8];
    char      qdir[2];
};
```

The fields in this header are:

nid Node ID of the system being backed up. For local files, this field contains the node ID of the local system.

qdir Full path name of the directory that is being backed up, beginning at the root directory.

File Headers

FS_INODE and **FS_NAME** headers have similar formats:

```
struct {
    struct hdr h;
    ushort      ino;
    ushort      mode;
    ushort      nlink;
    ushort      uid;
    ushort      gid;
    off_t       size;
    time_t      atime;
    time_t      mtime;
    time_t      ctime;
};
```

```

        ushort    devmaj;
        ushort    devmin;
        ushort    rdevmaj;
        ushort    rdevmin;
        off_t      dsize;
        char       name[4];
    };

```

The fields **mode** through **ctime** are copied from the i-node on disk.

Other fields are:

ino	I-number of file.
devmaj	Major device number of file system containing this file.
devmin	Minor device number of file system containing this file.
rdevmaj	Major device number of this file (character- and block-special files only).
rdevmin	Minor device number of this file (character- and block-special files only).
dsize	Size of the file after backup. This differs from size if the file was compressed during backup.
name	The null-terminated name of the file that is supplied by the user. This field is absent from FS_INODE headers.

End of Volume or Backup

FS_VOLEND and **FS_END** headers contain only the **hdr** structure.

Backup History

A backup history is kept in the **/etc/budate** file. The entries are in no particular order. Each entry has the following format:

```

struct {
    char    id_name[16];
    char    id_incno;
    time_t  id_budate;
};

```


backup

The fields of each entry are:

id_name	Name of the file system
id_incno	Incremental level number (0-9)
id_budate	Date of most recent backup of the file system at that level.

Related Information

In this book: “filesystems” on page 3-74.

The **backup**, **pack**, and **restore** commands in *AIX Operating System Commands Reference*.

cc.cfg

Purpose

Defines values used by the C compiler.

Description

The **cc.cfg** file defines values used by the **cc** program to run compilers. Normally, the **cc.cfg** file contains entries only for the C compiler provided with the system. Entries are made to this file to support C compilers for other systems as they are added.

This file is an attribute file. The name you specify when you run the **cc** program (it can be linked to several different names) determines which stanza of the **cc.cfg** file is used. Normally, the **cc** program runs as **cc**; therefore, the first stanza is almost always selected. If the **scc** program (stand-alone C compiler) is run, then the **scc** stanza is selected. If the **fcc** program (floating point) is selected, then the **fcc** stanza is selected. If the **vcc** program (**a.out-to-toc** conversion) is selected, then the **vcc** stanza is selected.

You can specify the following attributes:

as	The path name to be used for the assembler.
asflags	A string of values, separated by commas, to be passed to the assembler.
asopt	A string naming optional flags that, if encountered on the cc command line, should be passed to the assembler. See description of the cppopt field.
ccom	The path name to be used for the compiler. For a one-program compiler, this is the only compiler program provided. For a two-program compiler, this is the parser for the front end (also known as c0).
ccomflags	A string of values, separated by commas, to be passed to the compiler.
ccomopt	A string naming optional flags that, if encountered on the cc command line, should be passed to the compiler. See ppopt .
cgen	The path name to be used for the code generator of a two-program compiler (also known as c1).
cgenflags	A string of values, separated by commas, to be passed to the code generator. If a one-program compiler is used, these are appended to ccomflags .
cgenopt	A string naming optional flags that, if encountered on the cc commands line, should be passed to the code generator. See ppopt .

copt	The path name to be used for the peephole optimizer of a compiler with an explicit peephole program (also known as c2).
coptflags	A string of values, separated by commas, to be passed to the peephole optimizer.
coptopt	A string naming optional flags that, if encountered on the cc command line, should be passed to the peephole optimizer. See cppopt .
cpp	The path name to be used for the preprocessor.
cppflags	A string of flags, separated by commas, to be passed to the preprocessor.
cppopt	A string naming optional flags that, if encountered on the cc command line, should be passed to the preprocessor. The string is formatted for getopt() subroutine, as a concatenation of flag letters, with a letter followed by a : (colon) if the corresponding flag takes a parameter.
crt, mcrt	The path name of the object file passed as the first parameter to the link editor. In the presence of the -p flag to cc , the mcrt value is used; otherwise the crt value is used. The defaults are /lib/crt0.o and /lib/mcrt0.o .
csuffix	The suffix for C source programs, default c .
hsuffix	A second suffix for C source (enabled by using the -h flag to the cc command), default h .
ld	The path name to be used for the link editor.
ldflags	A string of values, separated by commas, to be passed to the link editor. These are in addition to those implicitly provided as described in the cc command.
ldopt	A string naming optional flags that, when encountered on the cc command line, to be passed to the link editor. See cppopt .
libraries	Flags, separated by commas, to be passed as the last parameters to the link editor as the default is libraries , the default is -lrts, -lc .
osuffix	The suffix for object files, the default is o .
ssuffix	The suffix for assembler programs, the default is s .
use	Values for attributes are taken from the named stanza in addition to the local stanza. For single-valued attributes, values in the use stanza apply if no value is provided in the local stanza (or default stanza). For comma-separated lists, the values from the use stanza are added to the values from the local stanza.

Example

* CC configuration file:

```
default:
    cpp      = /lib/cpp
```

* standard cc

```
cc:
    use      = DEFLT
    crt      = /lib/crt0.o
    mcrt     = /lib/mcrt0.o
    libraries = -lrts,-lc
    ldflags  = -n,-T0x10000000,-K
```

* direct floating point accelerator cc

```
fcc:
    use      = DEFLT
    crt      = /lib/crt0.o
    mcrt     = /lib/mcrt0.o
    libraries = -lrts,-lfm,-lfc
    ccomflags = -f
    ldflags  = -n,-T0x10000000,-K
```

* standard standalone cc

```
scc:
    use      = DEFLT
    crt      = /lib/crt2.o
    cppflags = -DSTANDALONE
    libraries = -l2
    ldflags  = -H4,-Y4
```

* atoc

```
vcc:
    ccom      = /lib/ccom
    ccomopt   = -O
    copt      = /lib/copt
    as        = /bin/as
```

cc.cfg

```
ld      = /bin/ld
cppflags = -Daiws,-DAIX
ldflags = -r,-X,-R4,-H4,-Y4,-T0x60
crt      = /usr/lib/vrmcrt.o
```

* common definitions

DEFLT:

```
ccom      = /lib/ccom
ccomopt    = 0f
copt       = /lib/copt
as         = /bin/as
ld         = /bin/ld
cppflags   = -Daiws,-DAIX
ldflags    = -e,start,-X
```

File

/etc/cc.cfg

Related Information

In this book: “getopt” on page 2-367 and “attributes” on page 3-20.

The **as**, **cc**, **ccp**, and **ld** commands in *AIX Operating System Commands Reference*.

config

Purpose

Describes system security configuration.

Synopsis

`security/config`

Description

The `/etc/security/config` attribute file consists of five stanzas for security-relevant programs. These stanzas are **audit**, **auditpr**, **passwd**, and **printer**.

The **audit** stanza of the `/etc/security/config` file contains the default configuration for system auditing. The **audit** command queries this stanza. Attributes are:

binmode The default audit trail mode. It may be **on**, **off**, or **panic**.

binsize The maximum size of each audit bin.

cmds The full path name of the audit backend command file.

The **password** stanza of the `/etc/security/config` file contains restrictions on new passwords supplied by users. The **newpass** command queries this information. Attributes are:

maxage The maximum number of weeks before the user must change a password.

minage The minimum number of weeks before the user can change a password.

minalpha The minimum number of alphabetic (a-z, A-Z) characters.

minother The minimum number of *other* (nonalphabetic) characters. For example, numbers or punctuation.

mindiff The minimum number of characters which must be different from the user's old password.

maxrepeat The maximum number of consecutive duplicate characters allowed in a password.

config

The **printer** stanza of **/etc/security/config** specifies printer labeling. The values of these attributes are strings with the escape sequence **%DAC**. Attributes in this stanza specify the label and the pages on which they print, as follows:

header The output header page.

trailer The output trailer page.

top The top of each output page.

bottom The bottom of each output page.

enabled Specifies whether printer labelling is on or off.

The **auditclasses** stanza of **/etc/security/config** contains a list of the audit classes and the audit events that comprise each. The list is in the form:

auditclasses:

general = login,su,passwd

audit = audit,auditevents,auditlog,auditproc

system = mount,umount,reboot,shutdown

tcb = TCBmod,TCBleak

dataxchg = restore,import

File

/etc/security/config

Related Information

In this book: “attributes” on page 3-20, and “audit” on page 2-31.

The **audit**, **auditpr**, **login**, **passwd**, **print**, and **stty** commands in *AIX Operating System Commands Reference*.

“Overview of International Character Support” in *Managing the AIX Operating System*.

connect.con

Purpose

Controls communication connections and data transfer.

Description

The connection configuration file, **/usr/lib/INnet/connect.con** or **\$HOME/bin/connect.con**, controls the setup of connections for the **connect** program and for certain optional communications programs. It provides a very general, flexible mechanism for specifying how connections are made and how data is transferred after making a connection.

The **connect.con** files are attribute files. The following attributes may appear in the connection control file.

Connection Options

The connection options and their descriptions are:

prefix, address, suffix

The telephone number to dial or the network address to contact. The actual number is constructed by concatenating the **prefix** (if any), the **address**, and the **suffix** (if any). Usually the **prefix** and **suffix** are defined in **/etc/ports** because they depend on the peculiarities of the dialer, and the **address** is defined in **connect.con**.

Multiple addresses can be specified by consecutive **address** assignment lines or by multiple **address** values separated by commas. The addresses are tried in the order given. To specify a comma as part of the command that is sent to the modem, enclose the entire **address** value in quotation marks.

connect Type of connection to make. This option is specified in **/etc/ports** since it is usually associated with the hardware configuration of the outgoing line. Permissible values are:

- | | |
|-------------------|---|
| permanent | The connection is hard-wired. No dialing or other special attention is needed. |
| manual | The connection must be completed manually. This generally implies a modem that does not dial; for example, an acoustic coupler. |
| hayes_1200 | The line has a Hayes Stack Smartmodem 1200. |

hayes_2400	The line has a Hayes Stack Smartmodem 2400.
vadic	The line has a Racal-Vadic 3451P autodialer.
ventel	The line has a Ventel MD212+ autodialer.
other_name	The line is associated with a dialer program, which is not built into the connect program. This option allows you to augment the capabilities of the connect program and other communications programs when dealing with new types of communications lines and dialers. The program searches for the named dialer program in /usr/lib/INnet/dialers or \$HOME/bin .

The assumptions made for dialer programs you supply are: the port to be used can be opened prior to dialing and the file will be opened as descriptor 3. Two parameters are passed: number to dial as parameter 1, and dialer hardware to use or value of the dialer option, if any as parameter 2. Any code exit from the dialer except 0 indicates the dialer failed. The failure code returned by the dialer determines the message printed by the programs.

linetype	Type of communication line protocols, either synchronous or asynchronous . Since different protocols are used on different line types, the talker programs can differ. The default linetype is asynchronous .
type	The name invoked with the connect program that determines the kind of connection attempted. Only those stanzas with the proper type are processed. Currently, the connect program itself uses only terminal type stanzas. The default type is terminal .
use	This option directs the connect program to read the named stanza and follow the instructions there.

Line Options and Parameters

Line options and parameters used are:

min	The minimum value to use in kernel buffering. Min value characters must be received before a call to the read system call returns, unless value specified in time elapses.
parity	The line is checked for the indicated parity: even , odd , any , or none .
speed	The transmission speed, generally 110, 300, 1200, 2400, 9600, and so on.

- time** The value to use in kernel buffering. **Time** in tenths of a second to receive a character before a call to the **read** system call returns unless **min** characters are received. See the discussion of ICANON in "termio" on page 5-133. Setting these parameters can result in improved performance.
- timeout** The time limit to complete the connection in seconds. When the time limit expires, the connection is aborted. This attribute is not needed for devices with a built-in time out.

System Options

The system options are:

- device** The name of the special file to use to make the connection. The device must appear in **/etc/ports** (see "ports" on page 3-173) and the information in the ports file entry that is made available to the **connect** program. Note that this attribute can appear only in the last of the list of stanzas associated with making the connection on this device, and that the **use** option must not appear.
- dialer** This option specifies the dialer hardware to be used in dialing the number. It is normally in **/etc/ports** file, associated with the device to be used. It may also be specified in a connection file, so that its value can be passed to a user-specified dialer program.

Diagnostics

The following diagnostics are displayed, based on the return value from system- or user-supplied dialer programs. The values 8 through 14 are treated as fatal errors.

Code	Message
0	Connected
1	Cannot open dialer
2	Busy or no answer
3	Not able to fork
4	Terminated attempts
5	Communication failure
6	Busy
7	No answer
8	Dead phone
9	Bad phone number
10	Cannot open device specified
11	Address not specified
12	Bad connect.con format
13	Cannot run dialer
14	No autodialer specified.

Login Script

A login script is file with the given name that is interpreted prior to notifying you that the connection is complete. Script files are located either in the **\$HOME/bin** file or in the **/usr/lib/INnet/scripts** file.

script A script file is organized into a group of states. In each state, the script file optionally sends a string to the remote system, then waits for a response. Several possible responses can be listed for each state along with an action to be performed if the response is received. A time limit can also be set in each state, along with an action to be performed if the time expires without an expected string arriving. The actions are to terminate script interpretations, with either a success or failure indication, or to move to another state. A sample script is shown under "Example" on page 3-41.

DONE

A successful termination of script interpretation.

ERROR *string*

An unsuccessful termination of script interpretation. The last message received from the remote site is reported to you.

GOTO *n*

Continues processing in state *n*.

RECV *string action*

This action is performed if the given string is received.

SEND *string*

Sends the given string to the remote system. Any name enclosed in braces in the string is taken to be an option reference and is replaced by the value of that option. If that option is not present in the list of stanzas, you are prompted for its value using the option name as the prompt. If a - (dash) precedes the name within the braces, the typed characters are not echoed. This is handy for including passwords as parameters in the script file without having them stored on the system. Thus, script files can be given parameters so that they can be used in different connection stanzas and by different users.

STATE *n*

Declares the beginning of state *n*.

TIMER *n action*

This action is performed if no expected string is received in *n* seconds.

Talker Program

A talker program handles the work of moving data across a connection. This program runs after a connection is established. The default talker for the **connect** program is **atalk**. You can override this and specify your own talker program.

talker This is name of the program to run when the connection is made. The conventions observed between the **connect** program and the talker are not complex: the connection is opened by the program as file descriptor 3. The only flag passed by **connect** to the talker program is:

-lockfile

Note: If the **-l** flag is present, the talker must remove the named *lockfile* to make the port available to other users.

flags This option passes flags (other than the above) to the talker program. This option is valid with both default or user-specified talkers.

Example

A typical script might be:

```
STATE 0  RECV User:                GOTO 1
          TIMER 10                 ERROR "No login"

STATE 1  SEND "{myname}\n"
          RECV Password:           GOTO 2
          RECV "Unknown:"          ERROR "Name unknown"
          TIMER 10                 ERROR "No password msg"

STATE 2  SEND "{-mypass}\n"
          RECV "$"                 DONE
          RECV Invalid             ERROR "Wrong password"
          TIMER 20                 ERROR "No prompt"
```

This script could be used for login to a remote RT system. In this file, **connect** waits up to 10 seconds for a **User:** prompt. When received, it sends the value of the *myname* option from the control file or by prompt, as the user name. It then waits for 10 seconds for the **Password:** prompt, then it sends the value of *mypass* as the password. The password is not echoed. It then waits another 20 seconds for another prompt. At each stage, it looks for messages that could occur if the given user name or password is invalid. With more states, you can write a script that tries several different user names and types the necessary information to dial through a port selector.

Files

`/usr/lib/INnet/connect.con`
`$HOME/bin/connect.con`

Related Information

In this book: “attributes” on page 3-20, “ports” on page 3-173, and “termio” on page 5-133.
The **connect** and **uucp** commands in *AIX Operating System Commands Reference*.
INmail/INnet/FTP.

core

Purpose

Contains an image of memory at the time of an error.

Synopsis

```
#include <core.h>
#include <sys/param.h>
#include <sys/reg.h>
#include <sys/user.h>
```

Description

The system writes a memory image of a terminated process when various errors occur in a **core** file in the current directory. See the **signal** system call for the list of errors. The most common are memory address violations, illegal instructions, bus errors, and user-generated quit signals. The memory image, called **core**, is written in the process working directory. A process with an effective user ID that is different from the real user ID does not produce a memory image.

The first section of the memory image is a copy of the system data per user process, including the contents of the registers as they exist at the time of the fault. The size of this section depends on the **usize** parameter defined in **/usr/include/sys/param.h**. The first section contains two parts. The first part is the user structure defined in **/usr/include/sys/user.h**. The second part is the process kernel stack. Note that the RT system stores the user process registers at the beginning of the user stack, instead of the end of the process kernel stack where they are normally stored on machines with stack, push, and pop instructions. The **/usr/include/sys/reg.h** structure outlines the long word offsets of the registers from the beginning of the user structure. The second section represents the actual contents of the user area when the image was written. If the text segment is separated from data space, it is not dumped.

File

core

Related Information

In this book: “setuid, setgid” on page 2-731 and “signal” on page 2-750.

The **crash** and **sdb** commands in *AIX Operating System Commands Reference*.

cpio

Purpose

Describes copy in and out (cpio) archive file.

Description

When the **-c** flag of the **cpio** command is not used, the header structure is:

```
struct {
    short
        h_magic,
        h_dev;
    unsigned short
        h_ino,
        h_mode,
        h_uid,
        h_gid;
    short
        h_nlink,
        h_rdev,
        h_mtime[2],
        h_namesize,
        h_filesize[2];
    char
        h_name[n];          /* described below */
} Hdr;
```

When the **cpio** command is used with the **-c** flag, the header for the **cpio** structure can be read as:

```
sscanf(Chdr,"%6ho%6ho%6ho%6ho%6ho%6ho%6ho%6ho%11lo%6ho%11lo%s",
&Hdr.h_magic, &Hdr.h_dev, &Hdr.h_ino, &Hdr.h_mode,
&Hdr.h_uid, &Hdr.h_gid, &Hdr.h_nlink, &Hdr.h_rdev,
&Longtime, &Hdr.h_namesize, &Longfile, &Hdr.h_name);
```

Longtime and **Longfile** are equivalent to **Hdr.h-mtime** and **Hdr.h-filesize**, respectively. The contents of each file together with other items describing the file are recorded in an element of the array of varying length structures. The member **h-magic** contains the constant octal 070707 (or 0x71c7). The **stat** system call explains the meaning of structure members **h-dev** through **h-mtime**. The length of the null-terminated path name, **h-name**, including the null byte is indicated by n , where $n = (\mathbf{h-namesize \% 2}) + \mathbf{h-namesize}$. In other words, n is equal to **h-namesize** if **h-namesize** is even. If **h-namesize** is odd, n is equal to **h-namesize** + 1.

The last record of the archive always contains the name **TRAILER!!!**. Special files, directories, and the trailer are recorded with **h-filesize** equal to 0.

Related Information

In this book: “scanf, fscanf, sscanf, NLscanf, NLfscanf, NLsscanf” on page 2-694 and “stat, fstat, lstat” on page 2-786.

The **cpio** and **find** commands in *AIX Operating System Commands Reference*.

ddi

Purpose

Contains device-dependent information (ddi).

Description

A **ddi** file contains information for customizing classes (or types) of hardware adapters or devices supported by the system. The information in this file can be modified using the **devices** command or an editor program. The **ddi** files are attribute files that are located in the **/etc/ddi** directory. See "attributes" on page 3-20 for the format of attribute files.

The equivalent of a **ddi** file can be created and added to the system. *Customize helper* programs convert the parameters in the files into a corresponding **Define-Device** structure, which is used by the VRM device driver. A **ddi** file contains the following information:

- Device-dependent information. This is a series of keywords whose values the user supplies when the device is defined.
- Instructions to the customize helper program for processing input parameters.
- Mapping information for the **ddi** structure.
- Bit field mapping information.

The use of extended characters in **ddi** files is not supported.

Keywords

The following keywords are used in the stanzas of device-dependent information files. These keywords describe attributes and settings for a particular device that can be changed to suit your device.

Note: An * (asterisk) identifies keywords that are not displayed to the user by the **devices** command.

Keyword	Description	Possible Choices
aa	Automatic Answering: Specifies whether the device supports communication auto answering. If the aa keyword is false and the protocol is dc or dr, data terminal ready and request to send are raised when the port is opened. If aa is true and the protocol is dc or dr, only dtr is raised until ring indicate is detected, at which time rts is raised.	true, false
adacode	Adapter Code: Full path name of the adapter load module.	
ae	Automatic Enable: Refers to the method by which a port is automatically enabled at system start time. When ae is true, the port is enabled when the system is restarted. When ae is false, the port is disabled. When set to share or delay, the port is enabled, with the proper locking to permit outgoing calls to originate from the RT system.	true = enable, false = not enabled For tty devices only: share = shared bidirectional use, delay = delay login hearld
alf	Automatic Line Feed: Does the printer have automatic line feed with carriage return?	true, false
ami	Adapter Microcode IOCN: The IOCN of the microcode module for an adapter that requires an IPL.	
appt	Application Type: The type of application that runs on the link.	3280, rje
ars	Aspect Ratio Support: Does the printer have a set aspect ratio control?	yes, no
*at	Adapter Type: Indicates the type of hardware adapter being used. Adapter values are discussed in the adpts stanza section in the /etc/predefined file.	See /etc/predefined
backs	Backspace Support: Does the printer have the ability to backspace (move print head backward while printing a line)?	yes, no

Keyword	Description	Possible Choices
bigs	Bit-Image Graphics Support: Does the printer have bit image graphics controls?	yes, no
biopa	First I/O Port Address: Refers to the hardware adapter address.	
bm	Bottom Margin (last line): Refers to the number of the last writing line.	1 - [length(in.) × lines/in.]
bpc	Bits Per Character: Refers to the number of bits per character that is used to transmit data from the RT system to the terminal or modem. This keyword is usually set to 7 or 8, and must match the terminal or modem setting. International Character Support requires a bpc value of 8.	5, 6, 7, 8
brea	Bus RAM End Address: If not zero, refers to the adapter's end RAM address on the I/O bus.	
brsa	Bus RAM Start Address: If not zero, refers to the adapter's start RAM address on the I/O bus.	
bs	Block Size: Refers to the size of sectors of storage on the fixed disk; changes in number of blocks on minidisks can affect system performance.	512, 1024, 2048
bsess	Base Session: Indicates the base session number for the link.	
cache	Default Cache: Full path name of the default cache.	
cachesetup	Cache Setup: Full path name of the setup job that is run after a user-defined cache is set up.	
cdp	Condensed Print: Does the printer support printing in condensed characters?	yes, no
cdpg	Code Page: Specifies the code page loaded into the printer.	437 (PC), 850 (MLP)

Keyword	Description	Possible Choices
cn	DMA Channel Number: Refers to DMA channel number.	0 - 9
colp	Color Printer: Refers to whether the printer is capable of printing in color.	yes, no
cps	Condensed Print Support: Can the printer print in condensed characters?	yes, no
cp1 - cp8	Code Page 1 through Code Page 8: Specifies the code pages loaded into the printer. The IBM 5201 Printer and 4201 Proprinter use only cp1 and cp2 .	PC, A, B, C, D, P0, P1, P2, MLP
cr	Color Ribbon: Does the printer have a color ribbon?	yes, no
cs	Character Set: Refers to complete groups of characters that a printer can support.	1, 2
cus	Continuous Underscore Support: Supports the escape - (minus) control.	yes, no
*ddbwn	Device Data Bus Width: Refers to the data bus width of the adapter.	8, 16
devname	Adapter Device Name: Refers to the device name of the communications hardware adapter to be used by the data link control.	token0, token1, mpd0, mpd1, mpd2, mpd3, net0, net1
*dmao	Uses DMA Support Only: Refers to whether the adapter supports DMA transfers.	true, false
*dmas	DMA Support: Refers to whether the adapter supports DMA transfers.	true, false
dnec	Do Not Enable DMA Channel: The channel is not enabled if the value is true .	true, false

Keyword	Description	Possible Choices
dpc	Default Print Color: Refers to the color to use for printing when a file does not contain codes that specify a color: usually black, blue, red, or yellow.	black, blue, red, yellow
ds	Data Stream: Postscript, Proprinter XL emulation, or self defining. (PostScript is assumed if the first character of the data stream is %; otherwise, Proprinter XL emulation is assumed.)	ps, ppxl, andy
dsp	Double Strike Print: Should double-strike be turned on?	yes, no
dsp	Double Strike Print Support: Does the printer have a control double-strike characters and provide boldface?	yes, no
dvam	Device Attachment Method: Refers to whether your device is attached locally with a cable or connected remotely through a modem. Opens to remotely attached devices do not complete until the Carrier Detect, Clear To Send, Data Set Ready, and Ring Indicate signals are set. Local devices, however, can be opened regardless of the setting of these signals.	0 = local, 1 = remote (modem)
dwp	Double Width Print: Should a file be printed with a double-width character set?	yes, no
dwps	Double Width Print Support: Does the printer have the ability to print with a double-width character set?	yes, no
ei	Enable Adapter Interrupts.	
*ei1 *ei2	Enable 1st / 2nd Interrupt Level: Refers to whether the adapter's interrupts are to be enabled when the adapter is initialized.	true, false
ep	Emphasized Print: Should emphasized print be turned on? Every character is overstruck with a second pass of the print head.	yes, no
eps	Emphasized Print Support: Does the printer have a control to do emphasized print?	yes, no

Keyword	Description	Possible Choices
eus	Expansion Unit Slot: Refers to whether the adapter resides in an IBM 6192 Expansion Unit.	true, false
fd	Fixed Disk: Refers to one of up to three circular plates used for storing data.	hdisk0, hdisk1, hdisk2
*fi	Frequency Input: Refers to the oscillator frequency used by the serial communications chips on the adapter.	< adapter dependent >
fid	Font ID: ID of the font used by the printer.	11
fl	Form (page) Length: Refers to the length of the paper in terms of the number of lines per page. The value is determined by multiplying the length of paper (in inches) by the number of lines printed per inch.	1 - [(in.) × lines/in.]
fnt1 - fnt8	Font 1 through Font 8: Specifies the printer fonts, such as letter gothic or prestige elite. This keyword is used for the IBM 3812 Pageprinter.	
fontdir	Font Directory Full path name (with trailing /) of the directory containing the font files.	
*fp	First Party DMA: Refers to whether the adapter has its own DMA controller.	true, false
fw	Form Width (right margin): Refers to the width of paper in terms of the number of characters per line. The value is determined by multiplying the width of the paper (in inches) by the number of characters printed per inch.	1 [width(in.) × pitch]
hsi	Horizontal Spacing Increment: What horizontal increment is used in the ESC K control?	60, 70, 120
hts	Horizontal Tab Support: Does the printer have horizontal tab controls?	yes, no
htvi	Text Vertical Increment: The vertical index increment used by subsequent CUU (ESC A) multibyte controls. (See "Multibyte Controls" on page 4-13.)	72

Keyword	Description	Possible Choices
*ic1 *ic2	Service Class of First or Second Interrupt: Refers to interrupt priority, where 0 is the highest.	0, 1, or 2
*il1 *il2	Interrupt Level Number of First or Second Interrupt: Refers to hardware adapter interrupt levels. These values must match the settings on the adapters with switch-selectable interrupt levels.	< adapter dependent >
iobd	Input/Output Bus Device.	true, false
*ioccb	Using DMA Four-Byte Buffering. Refers to whether adapters use DMA 4-byte buffering. ioccb must be false for current async adapters.	true, false
iof1 - iof8	Read/Write Flag for I/O Operation 1 through 8 : Is a read or write required to the corresponding I/O port address (pa1 - pa8)?	0 = Input, 1 = Output
*iopar	Number of I/O Port Addresses: Refers to the number of consecutive I/O addresses used by the adapter.	< adapter dependent >
iow1 - iow8	I/O Width for I/O Operation 1 through 8: Refers to the number of bits to be written to or read from the port address (pa1-pa8).	0 = 8 bit, 1 = 16 bit
ip	Initialize Printer: Refers to the initial state of the printer after power is applied.	true, false
ixp	Include Xon/Xoff Protocol: Refers to whether communication protocol includes Xon/Xoff flow control. If ixp is set to false, the protocol is not included. When set to true, Xon/Xoff flow control is used on both the received and transmitted data streams. The actual characters that are used to implement flow control are determined by the values of the roffv , ronv , toffv and tonv keywords.	true, false
js	Justification Support: Refers to printing with the right margin aligned.	yes, no

Keyword	Description	Possible Choices
kpoe	Keep Printing on Error: Should the printer complete the print job despite errors (without sending an error message to the user)?	yes, no
llo	Leave DTR and RTS Lines On.	true, false
lm	Left Margin: Refers to the area on a page between the left edge and the leftmost character position on the page.	0 - [width(in.) × pitch]
lobibp	Length of Buffers in Buffer Pool: The length in bytes of each buffer in the buffer pool of the Block I/O Communication Area (BIOCA).	
logger	PTY Supports Login Shell. Refers to whether the pty device supports a login shell. When set to true, the stanza for this pty is put into the <i>/etc/ports</i> file, which allows the port to be enabled and disabled.	true, false
lpi	Lines Per Inch: Refers to the number of print lines per inch, to line spacing density, and to the distance paper moves during a line feed.	6, 8
lrnc	Left/Right Margin Controls: Does the printer have the ability to change left and right margins (does it have left and right margin control codes)?	yes, no
lun	Logical Unit Number: Number associated with an addressable physical or logical device.	0 - 7
*mask	Mask: Refers to whether parameters should be changed and which should be reset to their default values. This keyword is usually set to F1FF.	F1FF
maxsaps	Maximum Service Access Points: Specifies how many concurrent service access points can open on the data link control.	1 - 126
maxbios	Maximum Number of Block I/O Devices: Refers to maximum number of block I/O devices.	1-100

Keyword	Description	Possible Choices
mc	Modem Controls: Refers to whether or not to enable modem controls.	true, false
mccs	Multibyte Control Code Support: If yes , then the printer supports IBMOEM multi-byte controls. If no , then the printer is assumed to function like an IBM 5152 printer.	yes, no
mnoal	Maximum Number of Attached LCCs: The maximum number of LCCs that can be attached to the device driver using the Block I/O Communication Area (BIOCA).	
mnonid	Maximum Number of Net IDs: The maximum number of network IDs that the device driver can support.	
nidd	Net ID Displacement: The offset (in bytes) into the receive data of the network ID.	
nidl	Net ID Length: The length in bytes of a network ID.	
nnfst	No Negotiate for Synchronous Transfer: Refers to whether the device negotiates for synchronous transfer.	true, false
noabb	Number of Allowed Bad Blocks.	
nob	Number of Blocks: Refers to the number of blocks in a minidisk.	
nobibp	Number of Buffers in Buffer Pool: The number of buffers to be allocated in the buffer pool of the Block I/O Communication Area (BIOCA).	
nobod	Number of Blocks on Device.	
nobodr	Number of Buffers on a Device Ring: The number of buffers to be allocated for each device ring queue in the Block I/O Communication Area (BIOCA).	
nobub	Number of 256-Byte Units/Block: Number of 256-byte units on each block.	

Keyword	Description	Possible Choices
*noi	Number of Interrupt Levels Used: Refers to the number of hardware interrupt levels used by the adapter.	< adapter dependent >
nops	Number of I/O Operations.	1 - 8
nor1 - nor8	Number of Repetitions for I/O Operation 1 through 8: Refers to the number of times the same I/O operation is performed to the corresponding port address (pa1 - pa8).	1 - 64
norbosr	Number of Receive Buffers on SLIH Ring queue.	
nosb	Number of Stop Bits: Refers to the number of stop bits used to frame each data character transmitted. The value chosen must be compatible with the setting of the bpc keyword. The value 1.5 can only be used for nosb when the bpc keyword is set to 5.	1, 1.5, 2
nospt	Number of Sectors per Track.	
*now	Number of DMA Sub-Channels. This keyword value should be set to 0 for all current async adapters.	0
*nr	No Read-Only Memory. Refers to whether the adapter contains Read-Only Memory (ROM).	true, false
nsess	Maximum Number of Sessions: The maximum number of sessions that can be run on the link.	1, 2, 3, 4, 5, 6, 7, 8
od1 - od8	Output Data for I/O Operation 1 through 8: Refers to the data to be written to the corresponding port address (pa1 - pa8) if the corresponding flag (iof1 - iof8) is set for output.	0000 - FFFF
om	Operation Mode: Refers to which communications operation mode is set. When set to half, half-duplex communication occurs using the modem control lines. This keyword is usually set to full, which allows data to flow in and out of the system at the same time.	half, full

Keyword	Description	Possible Choices
pa1 - pa8	Port Address for I/O Operation 1 through 8: Refers to the adapter port address being written to or read from to disable the adapter.	0000 - FFFF
pacs	Print All Characters Support: Does the printer support ESC ^ and ESC \ controls?	yes, no
pdt	Peripheral Device Type.	adapter, tape, disk
ph	Paper Handling: Refers to the way the printer handles different types of paper. The manual-feed printer stops at the end of each page and waits for the user to insert another sheet and press the start button. A printer with an automatic sheet-feed mechanism feeds paper to the printer.	0 = manual; 1 = automatic; 2 = continuous form paper.
pinit	User-Supplied Sequence: Refers to the control sequence the coprocessor uses to reset the printers whose fonts can be changed.	
pitch1 - pitch8	Character Pitch 1 through Character Pitch 8: Refers to the number of characters per linear inch; for instance, 10-pitch type has 10 characters per inch.	10, 12, 15
plot	Pass Data Directly to Device Without Modification.	yes, no
*pn	Port Number on Adapter: Refers to the hardware adapter port on multiport adapters.	0 - 8
pq	Print Quality: Can select (on some printers) degrees of print quality: dp (for fast, low quality), text (for better draft quality), letter (for high-quality final text).	dp, text, letter
prin	Printer Type: 0 = unspecified (functionally 5152); 1 = IBM 5152; 2 = IBM 5182; 3 = reserved; 4 = IBM 5201 Printer; 5 = IBM 4201 Proprinter; 6 = IBM 4202; 7 = IBM 3852.	0, 1, 2, 3, 4, 5, 6, 7

Keyword	Description	Possible Choices
printer	<p>Printer Configuration: You can configure the printer or terminal for each available session. The default value (0) is terminal across all sessions. To specify the printer, the values are as follows: 1 = First Session, 2 = Second Session, 3 = Third Session, 4 = Fourth Session, 5 = Fifth Session, 6 = Sixth Session (if allowed), 7 = Seventh Session (if allowed), and 8 = Eighth Session (if allowed).</p> <p>Note: For printer configuration to work both the host and the control unit must have the same level of support.</p>	0 = Terminal, 1 through 8 = Printer
pro	<p>Protocol: Refers to communication protocol, which determines how the modem control lines are used during a communications session. The pro keyword is generally set to dtr. The dc (Direct Connect) value allows attachment of devices that use hard-wired flow control.</p> <p>Note: While hard-wired flow control is sometimes referred to as DTR pacing, the pro keyword must be set to dc to support this function.</p> <p>When the pro keyword is set to cdstl and the aa (auto answer) keyword is set to true, the modem must send a Data Set Ready signal before the Data Terminal Ready signal is sent back to the modem.</p>	dtr, cdstl, dc
psd	Paper Source Drawer: Refers to the location of the paper drawer from which paper is drawn for printing.	1 = top; 2 = bottom
pssetup	PostScript Setup: Full path name of the file containing the PostScript setup job.	
pss	Proportional Spacing Support: Does the printer support proportionally spaced printing?	yes, no
pt	Parity Type: Refers to communication character parity, if any, that the transmitted data has. Received data is checked to ensure the proper parity. The parity types <i>mark</i> and <i>space</i> are not supported by the serial ports on the system board of the Model 6150.	even, odd, mark, space, none

Keyword	Description	Possible Choices
rdto	Receive Data Transfer Offset: The device driver using block I/O transfers the receive packet beginning at this offset into the buffer.	
rea	Bus ROM End Address: If not zero, refers to the adapter's start ROM address on the I/O bus.	
rl	RAS Length: The length in words of the RAS section of the Define Device structure.	
rlfs	Reverse Line Feed Support: Does the printer support the ESC J control?	yes, no
roffv	Receive Xoff Value: Refers to the character sent by the RT system to instruct the remote device to stop sending data (when the ixp keyword is true.) This value is usually set to 0x13.	00 - FF
ronv	Receive Xon Value: Refers to the character sent by the RT system to instruct the remote device to resume sending data (when the ixp keyword is true.) This value is usually set to 0x11.	00 - FF
rsa	Bus ROM Start Address: If not zero, refers to the adapter's end ROM address on the I/O bus.	
rtrig	Receive Buffer Trigger: If the buffer has receive data buffering capability rtrig selects the number of bytes that should accumulate before the adapter interrupts the processor. This value is not used by the serial ports on the system board of the model 6150.	1, 4, 8, 14
rts	Receive/Transmit Speed: Refers to the communication baud rate.	50, 75, 110, 134.5, 150, 300, 600, 1200, 1800, 2000, 2400, 3600, 4800, 7200, 9600, 19200

Keyword	Description	Possible Choices
rxringq	Receive Ring Queue Size: Specifies the number of receive buffers that can be queued for each data link control service access point.	1 - 255
*rxt	Receive Xoff Threshold: Refers to the minimum amount of free buffer space on the adapter. If the free space falls below the amount specified, the RT system sends the Receive Xoff character (roffv) to the attached device (if the keyword ixp is true). After an Xoff has been sent, all received data must be processed before the RT system sends the Receive Xon character (ronv).	< default = 0x14 >
sa	Strobe Active.	true, false
*sdmac	Shared DMA Channel: Refers to whether a hardware adapter can share the DMA channel.	true, false
*sg	DMA Scatter/Gather Support: Refers to whether the adapter provides the capability to scatter and gather I/O data that has gone through DMA.	true, false
*si1 *si2	Share First or Second Interrupt Levels: Refers to whether the adapter can share the hardware interrupt levels 1 or 2.	true, false
sid	SCSI ID: Refers to the SCSI ID number.	0 - 6
slap	Skip Lines at Perforation: Refers to the number of lines skipped at page breaks. The number is divided by two, so that half the blank lines appear at the bottom of one page and half at the top of the next.	0-[length(in.) × lines/in.]

Keyword	Description	Possible Choices
slow	<p>Slow Device Support: Refers to whether DFT slow device support is enabled. You can specify slow device on a session by session basis. The default is disabled (0) for each session. The values to enable a session are: 1 = First Session, 2 = Second Session, 3 = Third Session, 4 = Fourth Session, 5 = Fifth Session, 6 = Sixth Session (if allowed), 7 = Seventh Session (if allowed), and 8 = Eighth Session (if allowed).</p> <p>Note: For slow device to work both the control unit and the host must provide the same level of support.</p>	0 = Disabled, 1 through 8 = Enabled
sn	Slot Number: Refers to the slot in which the adapter is installed.	1 - 8
sns	Switched/Nonswitched: Refers to the state of the communication line connection. A value of true indicates that the connection uses a switched phone network.	true, false
sp	Select Printer.	true, false
sppt	Serial/Parallel Printer Type: Refers to whether the printer is a serial or parallel type.	1 = Parallel, 2 = Serial
srbt	SLIH Ring Buffer Threshold: The number of SLIH ring queue buffers that the device driver can use before requesting additional buffers from the block I/O device manager.	
sss	Superscript/Subscript Support: Does the printer have the ability to print in superscript and subscript mode?	yes, no
sysadd	<p>Specifies the action that the devices command takes after adding the device. The valid choices are:</p> <p>a Rebuilds the kernel and IPLs the system</p> <p>v Runs the vrconfig command</p> <p>none Takes no special action.</p>	a, v, none

Keyword	Description	Possible Choices
sysdel	Specifies the action that the devices takes after deleting the device. The valid choices are: a Rebuilds the kernel and IPLs the system v Runs the vrnconfig command none Takes no special action.	a, v, none
tbc	Transmit Buffer Count: Refers to the maximum number of bytes buffered on the adapter for transmission. Some devices can be overrun if this value is too high, since transmission of data staged at the adapter cannot be halted.	0x00 - 0x10
tm	Top Margin: Refers to the number of lines to be skipped at the top of a page before printing begins. If the user specifies 6 lines, the first print line will be line 7. The value is determined by the length of paper (in inches) multiplied by the number of lines per inch.	0 - [length(in.) × lines/in.]
toffv	Transmit Xoff Value: Refers to a character against which all incoming data is compared. If the ixp keyword is true and this character is received, the RT system stops transmitting as soon as possible. The value is usually set to 0x13.	0x00 - 0xFF
tonv	Transmit Xon Value: Refers to a character against which all incoming data is compared. If the ixp keyword is true and this character is received, the RT system resumes sending data. The value is usually set to 0x11. The value 0xFF indicates that any character received after the toffv character should be interpreted as a tonv character.	0x00 - 0xFF
tt	Terminal Type: Refers to the type of device attached.	
txringq	Transmit Ring Queue Size: Specifies the number of transmit buffers that can be queued for the data link control at one time.	1 - 255
type1 - type8	Typestyle 1 through Typestyle 8: Refers to a typestyle such as bold or italic.	

Keyword	Description	Possible Choices
urpim	User to Receive Printer Intervention Messages: Refers to whether printer intervention messages are sent to any valid user or to the user who queued the print job.	Any user ID, pjo = Print Job Owner
vhs	Variable Horizontal Spacing: Does the printer have ESC d and ESC e controls?	yes, no
vpqs	Variable Print Quality Support: Does the printer have the ability to print different degrees of quality?	yes, no
vsi	Vertical Spacing Increment: Refers to parts of inch supported in ESC 3 and ESC J controls.	216, 144
vtS	Vertical Tab Support: Does the printer support vertical tabs?	yes, no
wll	Wrap Long Lines: Does the printer "wrap" lines? That is, will it break lines longer than the specified form width at the right margin and print the remainder on the next line?	yes, no
12ps	12 Pitch Support: Does the printer support 12 pitch?	yes, no

Files

```

/etc/ddi/diskette
/etc/ddi/enet
/etc/ddi/float
/etc/ddi/font
/etc/ddi/opprinter
/etc/ddi/plotter
/etc/ddi/pprinter
/etc/ddi/sprinter
/etc/ddi/tty

```

Related Information

In this book: “attributes” on page 3-20, “descriptions” on page 3-65, “kaf” on page 3-121, “master” on page 3-125, “options” on page 3-165, “predefined” on page 3-180, and “system” on page 3-210.

The **devices** command in *AIX Operating System Commands Reference*.

descriptions

Purpose

Describes the meaning of **ddi** file keywords.

Description

The **/etc/ddi/descriptions** file contains a sorted list of descriptions for each of the keywords used in **ddi** files. The **devices** command uses this file to explain the meanings of the keywords during the **add**, **change**, and **showdev** subcommands.

The **/etc/ddi/descriptions** file must be sorted by keyword, and each line must follow the following format:

keyword description

where:

<i>keyword</i>	Names a keyword that is used in a ddi file. This field is exactly 10 characters long, is padded on the right with spaces, and contains no tabs.
<i>description</i>	Describes the meaning of the keyword. This field is exactly 28 characters long, is padded on the right with spaces, and contains no tab characters.

Note: The **/etc/ddi/descriptions** file must be sorted alphabetically by the *keyword* field. If it is not sorted, then the **devices** commands displays incorrect information about the meanings of keywords.

The use of extended characters in the **/etc/ddi/descriptions** file is not supported.

File

/etc/ddi/descriptions

Related Information

In this book: “ddi” on page 3-47 and “options” on page 3-165.

devinfo

devinfo

Purpose

Contains device characteristics.

Synopsis

```
#include <sys/devinfo.h>
```

Description

The **devinfo** structure is defined for each device. The **IOCINFO** operation of the **ioctl** system call fills in this structure. The information returned by a device varies. Most devices, other than disk devices, return a **devtype** value and the remainder of this structure contains zeros. This structure provides information about the capabilities of a device, rather than its current status or settings. For example, types of information provided are the number of characters a printer handles per line or the diskette capacity in number of blocks.

The maximum size of this structure is 12 bytes (no longer than the disk version), so that programs can use the **ioctl** system call without concern of overrun due to increasing size.

```
struct devinfo
{ char devtype;
  char flags;
  union
  {
    struct                /* for disks */
    { short bytpsec; /* bytes per sector */
      short secptrk; /* sectors per track */
      short trkpcyl; /* tracks per cylinder */
      long numblks; /* blocks this minidisk */
    } dk;
    struct                /* for memory mapped displays */
    {
      char capab; /* capabilities */
      char mode; /* current mode */
    }
  }
};
```

```

        short hres;    /* horizontal resolution */
        short vres;    /* vertical resolution */
    } tt;
} un;
};

```

The following flags specify some generic capabilities (see **DD_DISK**):

Constant	Value	Function
DF_FIXED	01	Not removable
DF_RAND	02	Random access possible
DF_FAST	04	A relative term

The **devinfo** structures are defined for the following devices (specified in the **devtype** field):

DD_DISK	Indicates a disk. This devtype is R . The driver determines the values. The fixed disk has flags DF_RAND DF_FIXED DF_FAST , while the diskette has flags DF_RAND (see “fd” on page 5-33 and “hd” on page 5-36). The number of the bytes per sector, sectors per track, and tracks per cylinder for the fixed disk are predetermined. The minidisk table determines the number of blocks. For the diskette, the minor device driver or the physical media determines this information when the device is opened.
DD_LP	Indicates a line printer. The devtype is l . This fills in the devtype field and returns zeros for the rest of the structure.
DD_PSEU	Indicates a pseudo-device. This devtype is Z .
DD_RTC	Indicates a real-time (calendar) clock. This devtype is c .
DD_TAPE	Indicates a magnetic tape. This devtype is M .
DD_TTY	Indicates a terminal. This returns a devtype of T and zeros for the rest of the structure.
DT_STREAM	Indicates a streaming tape drive. The devtype is 2 .
DT_STRTSTP	Indicates a start-stop tape drive. The devtype is 2 .

Related Information

In this book: “ioctl” on page 2-407, “fd” on page 5-33, and “hd” on page 5-36.

dir

Purpose

Describes the format of a directory.

Synopsis

```
#include <sys/dir.h>
```

Description

A directory is a file that a user is not allowed to write into directly. A directory file contains a 16-byte entry for each file in it. A bit in the flag word of the i-node entry indicates that the corresponding file should be treated as a directory. For additional information about a system volume format, see the “fs” on page 3-100. The structure of a directory entry as given in the include file is:

```
#include <sys/types.h>
#ifndef DIRSIZ
#define DIRSIZ 14
#endif
struct direct
{
    ino_t    d_ino;
    char     d_name[DIRSIZ];
};
```

By convention, the first two entries in each directory are . (dot) and .. (dot dot). The first . is an entry for the directory itself. The .. entry is for the parent directory. The meaning of the .. entry for the root directory of the master file system is modified. There is no parent directory; therefore, the .. entry has the same meaning as the . entry.

Related Information

In this book: “dirent.h” on page 4-24, “fs” on page 3-100, and “inode” on page 3-119.

errfile

Purpose

Contains system event log.

Synopsis

```
#include <sys/erec.h>
```

Description

When a system event occurs and logging is active, it generates an event record and passes the record to the event-logging daemon to be recorded in the event log. The **/etc/rasconf** file specifies the files where the events are to be logged. The default event log file is **/usr/adm/ras/errfile**.

Every record has a header. See “error” on page 5-31 for the structure of a header. Each type of event record has its own format. The **/usr/include/sys/erec.h** file shows the format of the events currently logged. The error daemon process gathers the records from memory and writes them in the files on disk. The event log file is opened (if existing) or created. Next, the process opens the **/dev/error** special file, formats and writes the nonvolatile random access memory (NVRAM), which can contain up to 16 bytes of information, and reads the events logged in memory. An analysis routine is called before an event is written to the **errfile**. For an error, this routine returns a buffer of probable cause information to aid in problem determination. This buffer is appended to the error entry, the length of the entry is adjusted, and then the entire entry is written to the file.

Some records in the event file are administrative. These include the startup record entered when logging is activated, the stop record written if the daemon is terminated gracefully, and the time-change record that accounts for changes in the system time of day.

Files

```
/usr/adm/errfile  
/dev/error  
/etc/rasconf
```


Related Information

In this book: “error” on page 5-31 and “rasconf” on page 3-189.
The **errdemon** in *AIX Operating System Commands Reference*.

events

Purpose

Specifies the tail format for audit events.

Synopsis

`/etc/security/audit/events`

Description

The `/etc/security/audit/events` file lists the audit events and their tail formats. This is an attribute file that contains only the **auditpr** stanza. This stanza contains auditable events and their specific tail formats.

File

`/etc/security/audit/events`

Related Information

In this book: “audit” on page 2-31, “auditlog” on page 2-39, and “audit” on page 3-22.

The **audit** and **auditpr** commands in *AIX Operating System Commands Reference*.

filesystems

Purpose

Centralizes file system characteristics.

Description

A file system is a complete directory structure, including a root directory and any directories and files beneath it. A file system is confined to a single partition. All of the information about the file system is centralized in the **filesystems** file. Most of the file system maintenance commands take their defaults from this file. The file is organized into stanzas whose names are file system names and whose contents are attribute-value pairs specifying characteristics of the file system.

The **filesystems** file serves two purposes:

- It documents the layout characteristics of the file systems.
- It frees the person who sets up the file system from having to enter and remember items such as the device where the file system resides because this information is defined in the file.

File System Attributes

Each stanza names the directory where the file system is normally mounted. The attributes specify all of the parameters of the file system. See “attributes” on page 3-20 for the format of an attribute file. The attributes currently used are:

account	Used by the dodisk command to determine the file systems to be processed by the accounting system. This value can be either TRUE or FALSE .
backupdev	Used by the backup and restore commands to determine the default output device associated with each file system. The value of this keyword is usually the name of a diskette or magnetic tape special file.
backuplen	Used by the backup command to determine the size of the default backup device associated with each file system. The size of a tape is measured in tracks times feet. For example, the backuplen for a 300-foot 9-track tape is 2700. This parameter is ignored for diskettes.
backuplev	Used by the backup command to determine the default backup level to take for each file system. Backup levels are discussed in the backup command.

boot	Used by the mkfs command to initialize the boot block of a new file system. This specifies the name of the load module to be placed into the first block of the file system.								
check	Used by the fsck command to determine the default file systems to be checked. TRUE enables checking while FALSE disables checking. If a number, rather than TRUE is specified, the file system is checked in the specified pass of checking. Multiple pass checking, described in fsck command in <i>AIX Operating System Commands Reference</i> , permits multiple file systems to be checked in parallel when multiple drives exist.								
cluster	Specifies the number of 512-byte disk blocks that the system treats as a unit. Only one or two values are supported. The RT default values are 4 for nonremovable disks and 1 for removable disks.								
cyl	Used by the mkfs command to initialize the free list and superblock of a new file system. The value is the number of blocks in one cylinder. It defines the size of an interleave cluster.								
dev	Identifies, for local mounts, either the block special file where the file system resides or the file or directory to be mounted. System management utilities use this attribute to map file system names to the corresponding device names. For remote mounts, identifies the file or directory to be mounted.								
free	Used by the df command to determine which file systems are to have their free space displayed by default. This value is either TRUE or FALSE .								
mount	Used by the mount command to determine whether or not this file system should be mounted by default. The possible values of mount are: <table> <tr> <td>automatic</td><td>Automatically mounts a file system when the system is rebooted. For example, in the sample file, the root file system line is mount = automatic. This means the root file system mounts automatically when the system is rebooted. The TRUE value is not used so that mount all does not try to mount it. Also, the value is not FALSE because certain utilities, such as ncheck, normally avoid file systems with a value of mount = FALSE.</td></tr> <tr> <td>false</td><td>This file system is not mounted by default.</td></tr> <tr> <td>inherit</td><td>When a remote file system is mounted, inherit mounts any additional file systems contained in the specified file system. This allows the local node to duplicate the file system structure of the server node, starting at the specified mount point.</td></tr> <tr> <td>readonly</td><td>When mount = FALSE, readonly is specified, the file system is mounted as read-only.</td></tr> </table>	automatic	Automatically mounts a file system when the system is rebooted. For example, in the sample file, the root file system line is mount = automatic . This means the root file system mounts automatically when the system is rebooted. The TRUE value is not used so that mount all does not try to mount it. Also, the value is not FALSE because certain utilities, such as ncheck , normally avoid file systems with a value of mount = FALSE .	false	This file system is not mounted by default.	inherit	When a remote file system is mounted, inherit mounts any additional file systems contained in the specified file system. This allows the local node to duplicate the file system structure of the server node, starting at the specified mount point.	readonly	When mount = FALSE , readonly is specified, the file system is mounted as read-only.
automatic	Automatically mounts a file system when the system is rebooted. For example, in the sample file, the root file system line is mount = automatic . This means the root file system mounts automatically when the system is rebooted. The TRUE value is not used so that mount all does not try to mount it. Also, the value is not FALSE because certain utilities, such as ncheck , normally avoid file systems with a value of mount = FALSE .								
false	This file system is not mounted by default.								
inherit	When a remote file system is mounted, inherit mounts any additional file systems contained in the specified file system. This allows the local node to duplicate the file system structure of the server node, starting at the specified mount point.								
readonly	When mount = FALSE , readonly is specified, the file system is mounted as read-only.								

removable	If mount = TRUE , removable is specified, a diskette file system is automatically mounted when its files are opened and unmounted when the opened files are closed. Also notice that in the example, this file is shipped designating two removable file systems, one having asterisks. The asterisks indicate commented lines in the file. The mkdir command must be used to create a directory in order to mount file system /dev/fd1 .
true	This file system is mounted by the mount all command.
nodename	Used by the mount command to determine which node contains the remote file system. If this attribute is not present, the mount is a local mount. The value of nodename can be either a valid node nickname or a valid node ID. This value can be overridden with the mount -n command.
size	Used by the mkfs command for reference and for building the file system. The value is the number of blocks in the file system.
skip	Used by the mkfs command to initialize the free list and superblock of a new file system. The value is the number of blocks to skip when the free list is interleaved. This number is processor- and device-specific.
type	Used by the mount command to determine whether or not this file system should be mounted. When the command mount -t string is issued, all of the currently unmounted file systems with a type equal to <i>string</i> are mounted.
vcheck	Used by the varyon command to determine which file systems to check. The value TRUE enables checking while FALSE disables checking.
vmount	Used by the varyon command to determine whether this file system should be mounted by default. The values of vmount are: TRUE , the varyon command mounts the file system, or FALSE , the file system is not mounted by default.
vol	Used by the mkfs command when initializing the label on a new file system. The value is a volume or pack label using a maximum of 6 characters. The file system label is always the stanza name.

Example

```

*
* File system information
*

default:
    vol      = "AIX"
    mount    = false
    check    = false
    free     = false
    backupdev = /dev/rfd0
    backuplen = 2400

/:
    dev      = /dev/hd0
    vol      = "root"
    mount    = automatic
    check    = true
    free     = true

/u:
    dev      = /dev/hd1
    vol      = "/u"
    mount    = true
    check    = true
    free     = true

/u/joe/1:
    dev      = /u/joe/1
    mount    = inherit
    nodename = vance

/usr:
    dev      = /dev/hd2
    vol      = "/usr"
    mount    = true
    check    = true

```


filesystems

```
        free      = true

/tmp:
    dev          = /dev/hd2
    vol          = "/tmp"
    mount        = true
    check        = true
    free         = true

/diskette0:
    dev          = /dev/fd0
    mount        = true,removable

* /diskette1:
*   dev          = /dev/fd1
*   mount        = true,removable
```

File

/etc/filesystems

Related Information

In this book: “attributes” on page 3-20 and “fs” on page 3-100.

The **backup**, **df**, **fsck**, **mkfs**, **mount**, **restore**, and **umount** commands in *AIX Operating System Commands Reference*.

fonts

Purpose

Defines character fonts for an **hft** display device in annotated, geometric, and rtfont formats.

Description

IBM supplies two sets of precompiled annotated text fonts with the AIX Operating System. One set of fonts is for the IBM 5081 Display Adapter and the other set is for all other devices supported by the Advanced Display Graphics Support Library (GSL). The fonts for the IBM 5081 Display Adapter cannot be used on other devices, and fonts for other devices cannot be used on the 5081 Display.

Annotated font definitions can be supplied to the VRM by configuring new font files into VRM. They can also be supplied by dynamically installing font modules into VRM and issuing an **HFRCONF** operation with the **ioctl** system call to inform VRM of the new fonts.

The provided annotated text fonts are automatically installed with the operating system. Other fonts that you create can be configured into the system by modifying the **/etc/master** file. You can select the active display font from the installed fonts by using the **display** command.

In addition to the precompiled fonts, IBM supplies the source for each non-5081 font, which you can copy and modify to create new font definitions. Since precompiled font files must be linked to the VRM at run time, the source font files must be compiled and converted to table of contents (TOC) format using the **vcc** and **vrmfnt** commands. For details about the TOC object module format, see *VRM Programming Support*. After converting the files, you must issue the **ioctl** call that does an **hft** reconfigure. (See “hft” on page 5-39.)

The GSL-supported devices also recognize two other font file formats: a geometric text format and an rtfont format. The format for geometric text fonts, also known as programmable character set (PCS) fonts, allows you to create fonts that can be scaled and rotated. The rtfont format allows you to design fonts that can be used with both X-Windows and the GSL. PCS font and rtfont files can be used on all GSL-supported devices including the IBM 5081 Display.

For more information on how to use the fonts described in this section, see Chapter 6, “Advanced Display Graphics Support Library.”

Annotated Text Font Format

An annotated text font definition file has three major parts in the following sequence:

- A header that describes the font. The header is the same for all annotated text fonts.
- A set of character descriptions:
 - 5081 fonts — A set of expanded character bit arrays that describes each character in the font.
 - Non-5081 fonts — A set of condensed raster mosaics that describes each character in the font.
- A lookup table that has an index entry to find each character representation in the font.
 - 5081 fonts — Lookup table entries are 16 bits.
 - Non-5081 fonts — Lookup table entries are 32 bits and each describes the start of its raster mosaics entry, its width, and the white space compressed from the top and bottom of its raster mosaics entry.

Annotated Text Font Header

The annotated text font header is a fixed-length structure common to all annotated text fonts for all displays. The VRM run-time binder uses the **DDDFSIZE** field in the header to link the font to the virtual terminal resource manager. The information in header fields is:

Offset in Bytes	Length in Bytes	Field	Description
0x00	4	DDDFSIZE	The size in bytes of the area containing the font and the lookup table.
0x04	2	fntclass	A number that uniquely identifies the format of the lookup table that follows: 0x01 = not a 5081 font 0x02 = a 5081 font
0x06	2	fntid	The name an application uses to identify a font. This must be a value within the range of 0 to 1024.
0x08	4	fntstyle	Font style.

Offset in Bytes	Length in Bytes	Field	Description
0x0C	4	fntattr	Identifies the attributes of the font. Possible values are: 0x0000 - no special values 0x0001 - bold version of this font 0x0002 - italic version of this font.
0x10	4	fnttotch	The total number of characters in the font. This is used to determine whether a specified character code is valid for this font.
0x14	4	fnttblsz	Total number of words in the font table.
0x18	2	fntbasln	The scan line within a character box of the baseline for characters in this font (zero origin).
0x1A	2	fntcapln	The scan line within a character box of the caps line for characters in this font (zero origin).
0x1C	2	fntcolumn	Width of character box in pels.
0x1E	2	fntrows	Height of character box in pels.
0x20	2	fntchrbit	Total number of bits per character.
0x22	2	fntultop	The scan line within the character box of the top line in the underscore (zero origin).
0x24	2	fntulbot	The scan line within the character box of the bottom line in the underscore (zero origin).
0x26	1	fntmonpt	Mono pitch flag in leftmost bit of this byte.
0x28	4	fntlkup	Byte offset from the beginning of this structure to the beginning of the font lookup table.

Annotated Text Font Raster Mosaics (non-5081)

This contains a definition for each character in the font. Each character is entered in this area with the horizontal slices bit-packed one right after the other. The first bit of the first character slice is forced to begin in the most significant bit of a byte. The raster mosaics start immediately after the header (0x2C from the start address of the font structure). See "Annotated Text Example One (non-5081)" on 3-83.

Annotated Text Character Bit Array (5081)

This contains a definition for each character in the 5081 font. Each character in the character bit array must be a multiple of 4 pels wide and a multiple of 4 pels high. Zeros are padded to the right and padded to the bottom of the character as needed to accomplish this.

Each 4x4 pel array is then stored in a 16-bit word with the first four bits of the array leftmost in the word and proceeding to the right.

The 4x4 arrays are stored beginning with the bottom left array in the character and are repeated across the bottom of the character. The process then continues at the left of the next higher horizontal row of 4x4 arrays and so on until the 4x4 array representing the top right corner of the character is stored in a 16-bit word. See "Annotated Text Example Two (5081)" on 3-85.

Annotated Text Font Lookup Table (non-5081)

The lookup table immediately follows the raster mosiac. There is one 32-bit lookup table entry for each character in the font. The lookup table can be found by adding the value **fontlkup** given in the header to the starting address of the font structure. The table entry for any given character is found by using the font position number as an index into the table. (See "display symbols" on page 4-26 for a list of the font position numbers.) Each lookup table entry contains the following fields:

Offset in Bits	Length in Bits	Field	Description
0	5	lkup-top	The number of blank scan lines that have been eliminated from the top of this character raster image (white space).
5	5	lkup-bot	The number of blank scan lines that have been eliminated from the bottom of this character raster image (white space).
10	6	lkup-width	Contains the width in pels of this particular character.
16	16	lkup-ref	Byte offset from the start of the the raster mosaics of the first scanline of the character's raster image.

Annotated Text Font Lookup Table (5081)

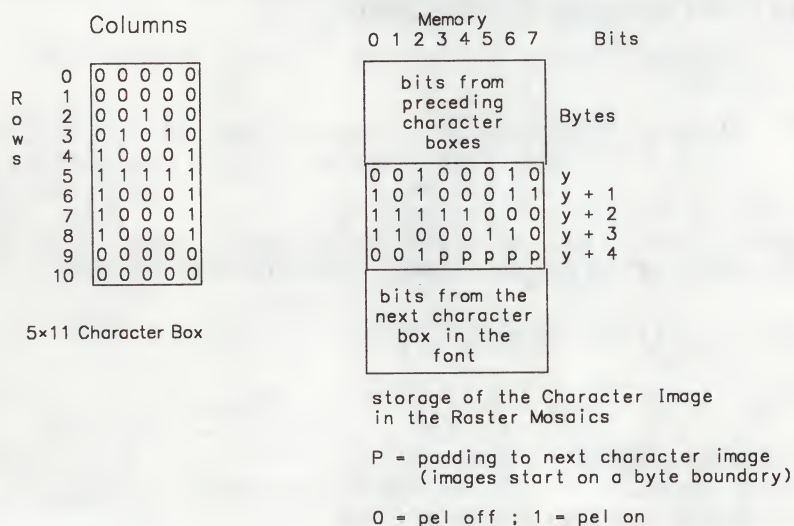
The character lookup table for IBM 5081 fonts contains an entry for each character or possible character in the font.

Each entry is 16 bits and contains the offset from the start of the character bit array to the first byte of the bit array for the corresponding character. That is, this offset is kept as a 16-bit word offset from the start of the bit array section. The character lookup table entry for any given character is found by concatenating an offset to the start of the code page in the character lookup table with the ASCII (or EBCDIC) character code, and adding the result to the starting address of the character lookup table found in the font header.

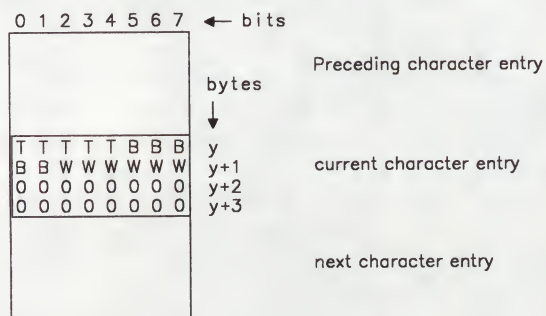
Annotated Text Example One (non-5081)

See Figure 3-1 on page 3-84 for this example. The character chosen is a capital A. This character is shown as it would appear on the display and how it would be stored in the raster mosaics. Also shown is the font lookup table entry for this character. Note that the data associated with the top and bottom two scan lines of the character image do not appear in the raster mosaics since they consist of zeros.

To reconstruct the character image from the raster mosaics, it is necessary to use the font lookup table. The display symbol code associated with the character that is to be displayed is used to access its corresponding 4-byte entry in the font lookup table. The information contained in a font lookup table entry is shown. The capital Ts represent the bits containing the number of top blank scan lines that were compressed from the character image. The capital Bs represent the bits containing the number of bottom blank scan lines that were compressed from the character image. The capital Ws represent the bits containing the width in pels of this character. Capital Os represent the bits containing the offset of the compressed portion of this character image data in the raster mosaics. For this example, the value associated with T is 2, the value associated with B is 2, and the width (W) is 5. The value associated with O is the offset of the yth byte of the raster mosaics.



Font Look-up Table



Font Look-Up Table Entry

Figure 3-1. Example of Annotated Text Font Storage (non-5081)

If this font is defined in a file named /usr/lib/vtm/nrm1.9x20s, then compile it and convert the a.out file to TOC format using the following commands:

```
vcc      /usr/lib/vtm/nrm1.9x20.s  -o nrm1.9x20.0
vrmfmt  nrm1.9x20.0  nrm1.9x20
```

Annotated Text Example Two (5081)

See Figure 3-2 on page 3-86 for this example. The character chosen is an uppercase A, and is shown as a 5x11 character box, which is then padded with zeros to the right and bottom to make the rows and columns a multiple of 4. The 4x4 arrays are then stored in 16-bit words beginning at the bottom left array in the box and continuing horizontally to the top right array in the character.

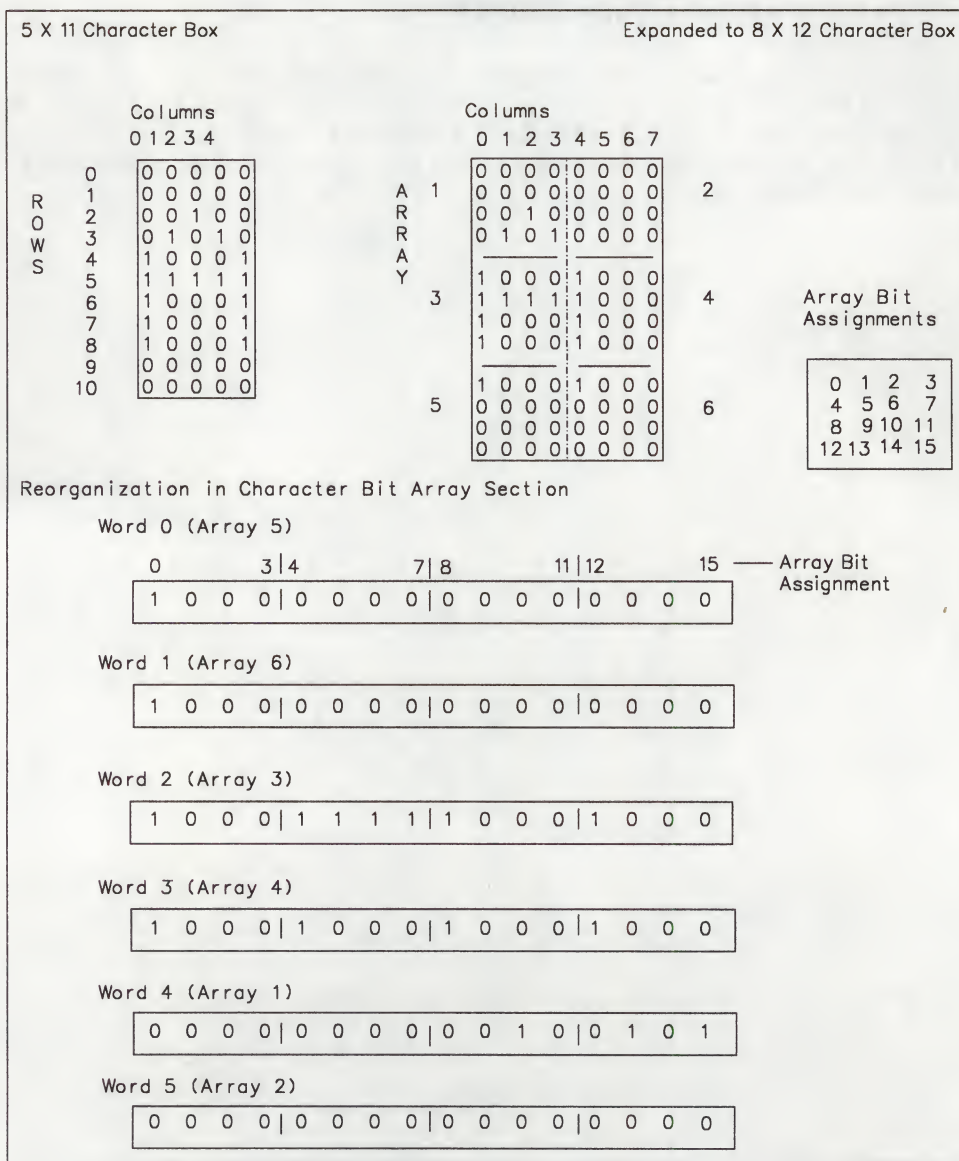


Figure 3-2. Example of Annotated Text Font Storage (5081)

Annotated Text Font Files (non-5081)

/etc/vtm/nrm1.4x8	Normal 4 by 8 micro font, compiled
/etc/vtm/nrm1.6x9	Normal 6 by 9 font, compiled
/etc/vtm/nrm1.6x11	Normal 6 by 11 font, compiled
/etc/vtm/nrm1.7x15	Normal 7 by 15 font, compiled
/etc/vtm/nrm1.7x22	Normal 7 by 22 font, compiled
/etc/vtm/nrm1.8x14	Normal 8 by 14 font, compiled
/etc/vtm/bld1.9x20	Bold 9 by 20 font, compiled
/etc/vtm/erg1.9x20	Ergonomic 9 by 20 font, compiled
/etc/vtm/itl1.9x20	Italic 9 by 20 font, compiled
/etc/vtm/nrm1.9x20	Normal 9 by 20 font, compiled
/etc/vtm/bld1.11x23	Bold 11 by 23 font, compiled
/etc/vtm/nrm1.11x23	Normal 11 by 23 font, compiled
/etc/vtm/nrm1.12x30	Normal 12 by 30 font, compiled
/etc/vtm/nrm1.18x40	Normal 18 by 40 title font, compiled
/usr/lib/vtm/nrm1.4x8.s	Normal 4 by 8 micro font, source
/usr/lib/vtm/nrm1.6x9.s	Normal 6 by 9 font, source
/usr/lib/vtm/nrm1.6x11.s	Normal 6 by 11 font, source
/usr/lib/vtm/nrm1.7x15.s	Normal 7 by 15 font, source
/usr/lib/vtm/nrm1.7x22.s	Normal 7 by 22 font, source
/usr/lib/vtm/nrm1.8x14.s	Normal 8 by 14 font, source
/usr/lib/vtm/bld1.9x20.s	Bold 9 by 20 font, source
/usr/lib/vtm/erg1.9x20.s	Ergonomic 9 by 20 font, source
/usr/lib/vtm/itl1.9x20.s	Italic 9 by 20 font, source
/usr/lib/vtm/nrm1.9x20.s	Normal 9 by 20 font, source
/usr/lib/vtm/bld1.11x23.s	Bold 11 by 23 font, source
/usr/lib/vtm/nrm1.11x23.s	Normal 11 by 23 font, source
/usr/lib/vtm/nrm1.12x30.s	Normal 12 by 30 font, source
/usr/lib/vtm/nrm1.18x40.s	Normal 18 by 40 title font, source.

Annotated Text Font Files (5081)

/etc/vtm/nrmMP1.4x8	Normal 4 by 8 micro font, compiled
/etc/vtm/nrmMP1.6x9	Normal 6 by 9 font, compiled
/etc/vtm/nrmMP1.6x11	Normal 6 by 11 font, compiled
/etc/vtm/nrmMP1.7x15	Normal 7 by 15 font, compiled
/etc/vtm/nrmMP1.7x22	Normal 7 by 22 font, compiled
/etc/vtm/nrmMP1.8x14	Normal 8 by 14 font, compiled
/etc/vtm/bldMP1.9x20	Bold 9 by 20 font, compiled
/etc/vtm/ergMP1.9x20	Ergonomic 9 by 20 font, compiled
/etc/vtm/itlMP1.9x20	Italic 9 by 20 font, compiled
/etc/vtm/nrmMP1.9x20	Normal 9 by 20 font, compiled
/etc/vtm/bldMP1.11x23	Bold 11 by 23 font, compiled
/etc/vtm/nrmMP1.11x23	Normal 11 by 23 font, compiled
/etc/vtm/nrmMP1.12x30	Normal 12 by 30 font, compiled
/etc/vtm/nrmMP1.18x40	Normal 18 by 40 title font, compiled.

Geometric Text Font Format

Geometric text fonts are also known as programmable character set (PCS) fonts and they can be used on all GSL supported devices including the IBM 5081 Display. Each character is defined as a series of moves or draws that define the shape of the character. The moves and draws are specified as X-Y pairs of signed relative values (relative to the previous ending point, or to the bottom left of the character box for the first X-Y pair). The range of the incremental values for the X and Y coordinates is -64 to +63.

Each character definition in the font consists of a 2-byte length field for the character definition followed by 2-byte X-Y entries:

Length of definition	2 bytes
sXXXXXX 1 sYYYYYY b	2 bytes
sXXXXXX 1 sYYYYYY b	2 bytes
sXXXXXX 1 sYYYYYY b	2 bytes
sXXXXXX 1 sYYYYYY b	2 bytes
· · · ·	
· · · ·	
· · · ·	
· · · ·	
sXXXXXX 1 sYYYYYY b	2 bytes

s is the sign bit (0 = positive, 1 = negative). Negative values are in twos complement notation.

b is the blanking bit. If b = 1, the primitive is blanked causing movement without display.

1 is the low order bit of the X coordinate field and must always be a 1.

If the first X-Y pair is a draw rather than a move, the line is drawn from the bottom left corner of the character box. A move is specified by the low-order bit of the Y coordinate being on. A draw is specified by the low-order bit being off. The last X-Y pair in the series for the character is defined by the length field.

Geometric Text Font Definition File

The PCS font definition file consists of:

- A header that contains identifier and control information
- A table of index values used to find each character definition
- The character definitions.

Offset in Bytes	Length in Bytes	Field	Description
0x00	2	length	The length of the PCS descriptor record including the length field.
0x02	4	Reserved	0x00000000
0x06	1		Bit 1 = 0 - EBCDIC = 1 - ASCII Bits 1-2 = Reserved Bits 3-7 = (Type) specifies the data format definition for programmable characters. One is defined: '00001'B = Type 1
0x07	1	Reserved	Must be 0.
0x08	2	fontid	This field identifies the programmable character set. Font IDs within the range of 1025 to 3267 are reserved for 1-byte character sets. IDs within the range of 32768 to 65535 are reserved for 2-byte character sets.
0x0A	1	segmentid	For 2-byte character sets, this byte contains the first byte of the 2-byte character code.
0x0B	1	Reserved	Must be 0.
0x0C	2	P	Range of X (between 0 and P)
0x0E	2	Q	Range of Y (between 0 and Q)
0x10	1	CP0	Starting character code within PCS (within the range of 0x21 to 0xFE.)
0x11	1	CPn	The last character code within this PCS. If CPn is 0, 0xFE is assumed. CPn must not be less than CP0.

fonts

Offset in Bytes	Length in Bytes	Field	Description
0x12	2	font baseline	The value of the font baseline in pixels in the Y direction from the bottom line of the character. This value is used in conjunction with the text alignment function.
0x14	2	font capline	The value of the font capline in pixels in the Y direction from the bottom line of the character. This value is used in conjunction with the text alignment function.
0x16	1	Reserved	
0x17	1	Default error code point	This is the character code within this PCS font that specifies the character to be displayed when an invalid code is encountered or the character code does not exist.
0x24	var	Character Index	This field contains 2-byte offsets to each character description. Each offset is from the beginning of the descriptor record.
var	var	Character Description	This field contains the character definitions, beginning with code point CP0, in ascending order.

P and Q together define the character box within which a normal character will fit. The values of P and Q are defined in device coordinate space (pixels) and control spacing between characters and new-line spacing. The bottom left corner of the box is 0,0 and the top right corner is P,Q. Characters can extend outside this box as P and Q control only the intercharacter spacing. You can override the value of P specified in the header by specifying a character inline spacing value greater than 0. Undefined character codes (outside the range CP0-CPn, or those with an index value of 0) are displayed as the default code point character.

Each character index value is the offset from the start of the header record to the actual character definition. The index must always be represented in its entirety, even if not all of the characters in the code range are defined. For example, the maximum length of the index, if CP0 is specified as 0x41 and CPn as 0xFF, is 191 multiplied by 2 bytes. For undefined characters, the index value should contain an offset to the default code definition.

Each character definition begins with a 2-byte length field which specifies the length of the character definition including the length field.

rtfont File Format

The font format described in this section, known as the *rtfont format*, is provided for use with X-Windows, Version 1.1 or higher and with the Advanced Display Graphics Support Library, Version 2.2 or higher. An rtfont definition file contains three major data structures in the following order:

1. A header structure, called **FONT-HEAD**, which contains information about the font in general. The header also contains offsets to the following two structures.
2. A character index array structure, called **CHAR-INDEX**, which contains modifiers of subsequent indices in the form of character offsets into the character data structure, also known as the character *glyphs*. The high-order four bits of each character index entry contain control information that defines how to interpret the 28 low-order bits of each entry.
3. A character glyph structure, called **CHAR-GLYPH**, which contains the character image data and information unique to each character.

Header Structure for rtfont Format

The header is a fixed-length structure common to all fonts in the rtfont format. This structure, called **FONT-HEAD**, is defined in the **rtfont.h** header file, and it contains the following members:

Offset in Bytes	Length in Bytes	Field	Description
0x00	16	font_id	The ID of the font.
0x10	4	off-to-index	The offset in bytes from the beginning of the file to the character index array.
0x14	4	off-to-glyphs	The offset in bytes from the beginning of the file to the character glyphs.
0x18	4	glyph-format	The format of the character image data.
0x1C	4	src-file-type	The type of font file from which this file was generated.
0x20	4	design-size	The design size in $(1/2)^{*}16$ points. This field is for use by T _E X, a typesetting system described in <i>The T_EXbook</i> , Donald E. Knuth, Addison Wesley Publishing Company, 1986. (See Note 1 on page 3-93.)

fonts

Offset in Bytes	Length in Bytes	Field	Description
0x24	4	check_sum	The checksum (T _E X). (See Note 1 on page 3-93.)
0x28	4	hppp	The number of horizontal pixels per point x 2**16. (See Note 1 on page 3-93.)
0x2C	4	vppp	The number of vertical pixels per point x 2**16. (See Note 1 on page 3-93.)
0x30	4	fiFlags	The flags field bits. (See Note 1 on page 3-93.)
0x34	4	firstCol	Reserved (set to 0). (See Note 1 on page 3-93.)
0x38	4	lastCol	Reserved (set to 0). (See Note 1 on page 3-93.)
0x3C	4	firstRow	Reserved (set to 0). (See Note 1 on page 3-93.)
0x40	4	lastRow	Reserved (set to 0). (See Note 1 on page 3-93.)
0x44	4	nProps	The number of properties. (See Note 1 on page 3-93.)
0x48	4	chDefault	The default character. (See Note 1 on page 3-93.)
0x4C	4	font_Descent	Extent below baseline for spacing. (See Note 1 on page 3-93.)
0x50	4	font_Ascent	The extent above baseline for spacing. (See Note 1 on page 3-93.)
0x54	20	minbounds	The minimum glyph metrics over all characters in font.
0x68	20	maxbounds	The maximum glyph metrics over all characters in font.
0x7C	4	pixDepth	The number of intensity bits per pixel. (See Note 1 on page 3-93.)

Offset in Bytes	Length in Bytes	Field	Description
0x80	4	glyphSets	The number of sets of glyphs. (See Note 1.)
0x84	4	raster_align	The scanline alignment for glyphs.
0x88	4	index_width	The width of each entry in the character index array.
0x8C	4	char_width	Fixed width of each character. (If set to 0, characters are variable width.)
0x90	4	char_height	Fixed height of each character. (If set to 0, characters are variable height.)
0x94	4	last_index	The last character index in the font.
0x98	4	codepoints_less_1	The number of code points minus 1.
0x9C	16	app_var	For application use.
0xAC	64	pad	Reserved (set to 0).

Notes:

1. This value is not supported by the Advanced Display Graphics Support Library or by the X-Windows Version 1.1 program.
2. This value is not supported by the X-Windows Version 1.1 program.

Some of the fields in the header structure require special values, as explained in the following list.

glyph-format This field has two possible values:

RASTER_DATA All glyphs are in raster format bit image.

PK_DATA All glyphs are in PK format. (See Note 1.)

src-file-type This field has one of the following values:

VRM_SRC_FILE The origin of this font file was a VRM fonts file, provided with the AIX Operating System.

PK_SRC_FILE The origin of this font file was a PK font file, a font structure generated by the METAFONT compiler. METAFONT is described in *The METAFONTbook*, Donald E. Knuth, Addison Wesley Publishing Company, 1986. For details on developing various font styles and sizes with METAFONT, refer to

Computer Modern Typefaces, Donald E. Knuth, Addison Wesley Publishing Company, 1986.

X10_SRC_FILE The origin of this font file was an X10 font file, distributed with the Massachusetts Institute of Technology X-Windows program, Version 10.4.

X11_SRC_FILE The origin of this font file was an X11 font file, distributed with the Massachusetts Institute of Technology X-Windows program, Version 11.

ORIG_SRC_FILE This is the original source file for this font.

raster-align

Raster format glyphs are stored beginning on a 32-bit boundary with scanlines packed from end to beginning. The first scanline of a raster format glyph is always on a 32-bit boundary. The significant scanline bits begin at the boundary of the scanline and continue for the width of the character. The boundary for subsequent scanlines is defined by the **raster-align** field as follows:

NO_ALIGN The first scanline is on a 32-bit boundary and subsequent scanlines for this character are bit packed. That is, the second scanline begins in the bit following the last bit of the first scanline, the third scanline follows the last bit of the second, and so on. (See Note 1 on page 3-93.)

BYTE_ALIGN The first scanline is on a 32-bit boundary and subsequent scanlines for this character begin on the next 8-bit boundary following the last bit of the previous scanline. (See Note 1 on page 3-93.)

HWD_ALIGN The first scanline is on a 32-bit boundary and subsequent scanlines for this character begin on the next 16-bit boundary following the last bit of the previous scanline.

FWD_ALIGN The first scanline is on a 32-bit boundary and subsequent scanlines for this character begin on the next 32-bit boundary following the last bit of the previous scanline. (See Note 2 on page 3-93.)

minbounds, maxbounds

These fields contain the maximum and minimum values of each individual **BOUNDS** structure for *all* characters in the font, as explained in the following section on the **CHAR_GLYPH** structure. The **minbounds.rbearing** value, for instance, must exist in at least one individual **BOUNDS** structure, and the **rbearing** field in any other individual **BOUNDS** structure must be greater than or equal to this value.

Character Index Array for rtfont Format

Since the data bytes within the data stream are used to access the character index array, the array must contain at least 256 entries. Entries for which a character is not defined should be set to the offset values of a valid default character. One such valid offset is zero. Since each font has at least one character defined, there is always a *first* character pointed to by offset zero, the first character in the glyph structure.

The character index information is contained in the **CHAR_INDEX** array, as defined in the **rtfont.h** header file. This structure contains the following elements:

Offset in Bytes	Length in Bytes	Field	Description
0x00	4	CHAR_INDEX	Character glyph offset or index modifier.

The low-order 28 bits of a character index array can be either an offset into the character glyph structure or a modifier value that modifies the next offset into the character index array. The high-order four bits of each entry define how the low-order 28 bits are interpreted. The value of the **CHAR_INDEX** field, when logically ANDed with the following values, is used to interpret the low-order 28 bits.

- INDBASE** Specifies a base modifier, such as ANSI SG0, SG1, and so on. (See Note 1 on page 3-93.)
- INDMOD** Specifies an index modifier, such as ANSI SS1, SS2, and so on.
- INDCPT** Indicates that the data stream source byte is not a code point.
- INDMASK** Specifies an offset to a glyph whose data stream source byte is a code point.

If the data source byte is not a code point and the low-order 28 bits do not indicate a base or index modifier, those bits *must* be a valid offset into the glyph structure.

Character Index Example

A sample data stream processing algorithm follows:

```
unsigned base_modifier, index_modifier, value, n;
    Both base_modifier and index_modifier are to be initialized
    to 0 for each independent character stream to be processed;

if ( (n=data_stream+index_modifier+base_modifier)<=FONT_HEAD.last_index)
{
    index_modifier=0;
    value=CHAR_INDEX[n]
    if (value & INDBASEUcl.
    {
        base_modifier=value & !INDMASK
        process next character in data stream
    }
    else if (value & INDMOD)
    {
        index_modifier=value & !INDMASK
        process next character in data stream
    }
    else
    {
        glyph_offset=value & !INDMASK
        process glyph
        process next character in data stream
    }
}
else
    offset into CHAR_INDEX is in error
```

Character Glyph Structure for rtfont Format

The character glyph structure contains information pertinent to each character in the font. The information per character is defined via the **CHAR_GLYPH** structure, as defined in the **rtfont.h** header file. This structure contains the following members:

Offset in Bytes	Length in Bytes	Field	Description
0x00	1	pk_format_flag	METAFONT information. (See Note 1 on page 3-93.)
0x02	3	resv	Reserved field (set to 0).
0x01	4	exists	Indicates whether or not a glyph exists.
0x04	4	tfm	T _E X font metric information. (See Note 1 on page 3-93.)
0x08	20	char_bounds	Character BOUNDS structure, defined below.
0x1C	var	char_bits	Character image.

Bounds Structure for rtfont Format

The **BOUNDS** structure, referenced by the **CHAR_GLYPH** structure above, is defined in the **rtfont.h** header file, and it contains the following members:

Offset in Bytes	Length in Bytes	Field	Description
0x00	4	rbearing	Character origin to right edge of raster. (See Note 2 on page 3-93.)
0x04	4	lbearing	Character origin to left edge of raster. (See Note 2 on page 3-93.)
0x08	4	descent	Baseline to bottom edge of raster. (See Note 2 on page 3-93.)
0x0C	4	ascent	Baseline to top edge of raster. (See Note 2 on page 3-93.)

Offset in Bytes	Length in Bytes	Field	Description
0x10	4	width	Advance to next character origin.

The glyph, or character data, can be drawn relative to any point in a given x,y coordinate system. The following description of the **BOUNDS** variables assumes the x,y position is on the baseline of the character. Coordinates are positive to the right and positive in the downward direction. The **pel box** is the area where the glyph is positioned on the screen when the rtfont is used.

The following diagram graphically portrays each of the fields in the **BOUNDS** structure and shows how these fields define the pel box, relative to the coordinates x and y. In this example:

1. The left vertical edge of the pel box is located at $x + lbearing$.
2. The right vertical edge of the pel box is located at $x + rbearing$.
3. The upper horizontal edge of the pel box is located at $y - ascent$.
4. The lower horizontal edge of the pel box is located at $y + descent$.
5. The origin for the next character is at the point $(x + width, y)$.
6. The width of the pel box, which defines the number of scan columns, is $rbearing - lbearing + 1$.
7. The height of the pel box, which defines the number of scan lines, is $ascent + descent + 1$.

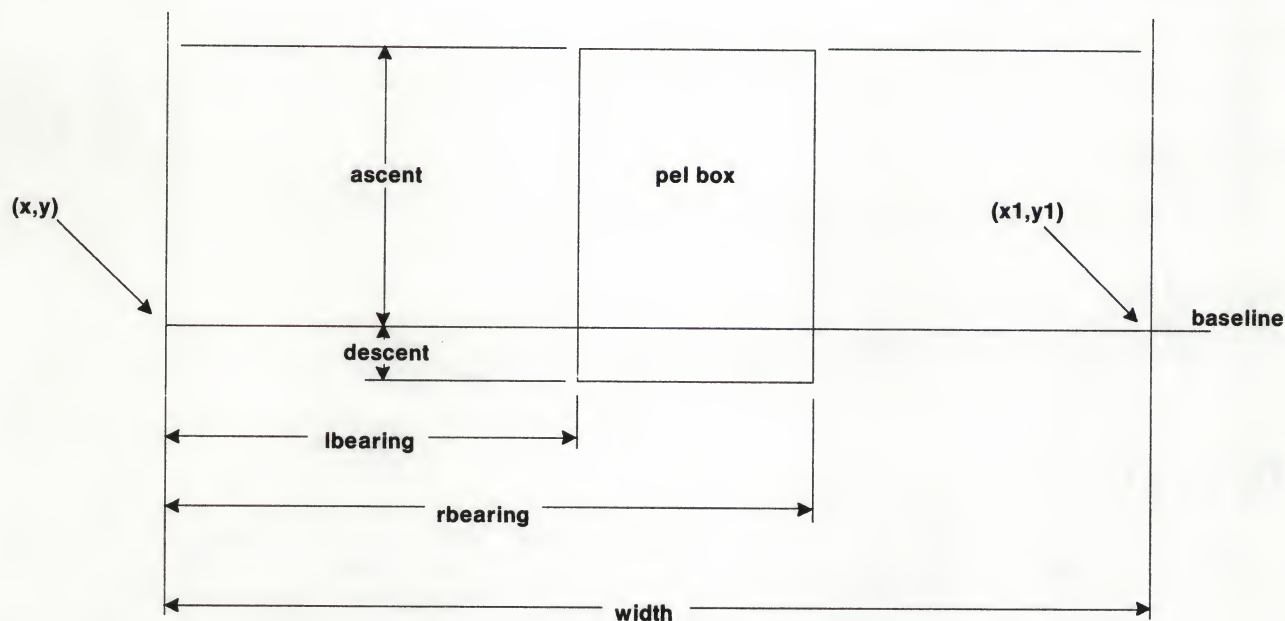


Figure 3-3. Example of an rtfon Pel Box

(x,y) is the position from which this character's pel box is referenced.
 $(x1,y1)$ is the position for the next character's pel box reference point.

Files

<code>/usr/include/rtfont.h</code>	Header file for the rtfon format.
<code>/usr/bin/vrm2rtfont</code>	Font conversion command.

Related Information

In this book: "master" on page 3-125, "data stream" on page 4-5, "display symbols" on page 4-26, "Reconfigure (HFRCONF)" on page 5-48, "gsgtat" on page 6-84, "gsgtxt" on page 6-89, "gstatt" on page 6-153, "gstext" on page 6-157, "gsxtat" on page 6-175, and "gsxtxt" on page 6-180.

The **display** command in *AIX Operating System Commands Reference*.

The discussion of the TOC object module format in *VRM Programming Support*.

Purpose

Contains the format of a file system volume.

Synopsis

```
#include <sys/types.h>
#include <sys/param.h>
#include <sys/filsys.h>
```

Description

A file system storage volume has a common format for certain vital information. A volume is divided into a number of logical blocks: 512-byte blocks for diskette and 2048-byte blocks for disks. The term **block** here refers to the unit of disk space allocation, which is some multiple of 512 bytes. The 512-byte unit is used to report or specify file sizes in all commands and subroutines, but here the term refers to a cluster of one or more such units. The RT system supports two similar but distinct file system formats, both of which are described by the following text.

The first format uses the byte order and integer size of the RT processing unit. The second format is compatible with the PC/IX file system format, which is based on the IBM PC-XT processing unit architecture. There are several differences between the two file system formats: the block size is 2048 bytes in the native file system format and 512 bytes in the other, and the definition of the superblock is different between the two. The number and order of bytes within multibyte data are different, and the processing units impose different restrictions on the alignment of 4-byte data. These last two differences affect fields within the superblock, i-node numbers within directories, and logical block numbers within i-nodes and indirect blocks.

The **mkfs** application makes a file system of the second format only when the file system device is a diskette.

Logical block 0 is unused and available to contain a bootstrap program or other information. Logical block 1 is the superblock.

The format of a native-format file system superblock follows.

```
#define      FSfixsz      112      / * Fixed-format region is 112 bytes long */

typedef      struct      filsys
{ /* basic file system parameters - initialized when FS is created */
    char      s_magic[4];      /* magic number: FSmagic = 0xdf817eb2 */
    char      s_flag[4];      /* flag word (see below) */
    daddr_t    s_fsize;      /* size in blocks of entire file system */
    ushort     s_bsize;      /* block size (in bytes) for this filsys */
    ushort     s_isize;      /* size (in blocks) of i-list and overhead */
    short      s_cyl;      /* number of blocks per cylinder */
    short      s_skip;      /* block interleaving factor */
    short      s_nicfree;      /* number of slots in block free list */
    short      s_nicino;      /* number of slots in free i-node list */
    short      s_sicfree;      /* byte offset to start of block free list */
    short      s_sicino;      /* byte offset to start of free i-node list */
    char      s_fname[6];      /* name of this file system */
    char      s_fpack[6];      /* name of this volume */
    short      s_nicfrag;      /* number of slots in fragment table */
    short      s_sicfrag;      /* byte offset to start of fragment table */
    daddr_t    s_swaplo;      /* start of swap area (currently unused) */
    daddr_t    s_nswap;      /* number of block swap area (currently unused) */
    char      s_rsvd[36];      /* reserved - must be zero */

    /* current file system state information, values change over time */

    ushort     s_tffrag;      /* number of fragmented files (currently unused) */
    ushort     s_tbfrag;      /* number of fragmented blocks (currently unused) */
    short      s_findex;      /* fragment allocation index (currently unused) */
    char      s_fmod;      /* superblock modified flag */
    char      s_ronly;      /* mounted read-only flag */
    daddr_t    s_tfree;      /* total free blocks */
    char      s_flock;      /* lock during free list manipulation */
    char      s_ilock;      /* lock during i-list manipulation */
    short      s_nfree;      /* number of addresses in s_free */
    ino_t      s_tinode;      /* total free inodes */
    short      s_ninode;      /* number of inodes in s_inode */
    time_t     s_time;      /* time of last superblock update */
}
```



```
/*
 * TOTAL LENGTH OF FIXED-FORMAT REGION: 112 bytes
 *
 * All variable length fields appear beyond this point, and are
 * described and pointed to by information in the fixed format
 * portion of the superblock:
 *
 *      daddr_t    s_free[s_nicfree]; <free block list>
 *      ino_t      s_inode[s_nicino]; <free I-node list>
 *      frag_t     s_frag[s_nicfrag]; <fragment table>
 *
 *      Macros defined below allow access to these tables.
 */

union                                /* Variable-format */
{ char    su_var [BSIZE-FSfixsz];
  struct
  {
      daddr_t su_free[NICFREE];
      ino_t   su_inode[NICINOD];
  } su_ovly;
} s_u;
} filsys_t;

#define s_free  s_u.su_ovly.su_free
#define s_inode s_u.su_ovly.su_inode
#define s_var   s_var

#define s_n      s_cyl           /* for compatibility with old systems */
#define s_m      s_skip
#define FSmagic "\337\201\176\262" /* octal magic number for file systems */
/* hex equivalent value for FSmagic number above is \df\81\7e\b2 */
#define s_cpu    s_flag[0]      /* Target cpu type code (same as in a.out files) */
#define s_type   s_flag[3]      /* File system type code (block size) */

#define Fs1b 1           /* 512 byte blocksize file system */
#define Fs2b 2           /* 1024 byte blocksize */
#define Fs4b 3           /* 2048 byte blocksize */
#define Fs8b 4           /* 4096 byte blocksize */
```

```
/*
 * Notes on s_fmod field:
 * This field is intended to be a three state flag with the third
 * state being a sticky state. The three states are:
 *
 * 0 = file system is clean and unmounted
 * 1 = file system is mounted
 * 2 = file system was mounted when dirty
 *
 * If you merely mount and unmount the file system, the flag
 * toggles back and forth between states 0 and 1. If you ever
 * mount the file system while the flag is in state 1 then it
 * goes to state 2 and stays there until you run fsck.
 * The only way to clean up a corrupted file system (and change
 * the flag from state 2 back to state 0) is to run fsck.
 * The bit above this tri-state (i.e. 04, FM_SDIRTY) is only used
 * in memory. It is never written to disk.
 */
#define FM_CLEAN    00 /* File system is clean and unmounted */
#define FM_MOUNT    01 /* File system is mounted cleanly */
#define FM_MDIRTY   02 /* File system was dirty when last mounted */
#define FM_SDIRTY   04 /* Superblock is dirty; this bit is not written */

#define FMOD(x)      ((x)==0?1:2)
#define FCLEAN(x)    ((x)==2?2:0)

/*
 * Macros for accessing elements in the variable-format region of the
 * superblock. "sbp" is a pointer to superblock, and "n" gives
 * the index of the element to be fetched.
 */

/* FREEino() -- Finds the nth element in the free I-node list. */
/* Each element of the free I-node list is of type "ino_t". */

#define FREEino(sbp,n) \
    (((ino_t *)((char *) (sbp) + (sbp) -> s_sicino))[n])
```

```
/* FREEblk() -- Finds the nth element in the free block list.  Each */
/*      element of the free block list is of type "daddr_t". */

#define FREEblk(sbp,n) \
    (((daddr_t *)((char *) (sbp) + (sbp) -> s_sicfree)) [n])

/* we have a NEW Format superblock */
#define _s_NEWF
    The format of a PC/IX-format file system superblock is:

/*
 *  Structure of the superblock
 */
struct    filsys
{
    ushort    s_isize;          /* size in blocks of i-list */
    daddr_t   s_fsize;          /* size in blocks of entire volume */
    short     s_nfree;          /* number of address in s_free */
    daddr_t   s_free[NICFREE];  /* free block list */
    short     s_ninode;         /* number of inodes in s_inode */
    ino_t     s_inode[NICINOD]; /* free I-node list */
    char      s_flock;          /* lock during free list manipulation */
    char      s_ilock;          /* lock during i-list manipulation */
    char      s_fmod;           /* superblock modified flag */
    char      s_ronly;          /* mounted read-only flag */
    time_t    s_time;           /* last superblock update */
    short     s_dinfo[4];       /* device information */
    daddr_t   s_tfree;          /* total free blocks */
    ino_t     s_tinode;         /* total free i-nodes */
    char      s_fname[6];       /* file system name */
    char      s_fpack[6];       /* file system pack name */
    long      s_fill[13];       /* fill out to 512 bytes */
    daddr_t   s_swaplo;         /* start of swap area */
    daddr_t   s_nswap;          /* number of blocks of swap */
    long      s_magic;          /* magic number for file systems */
    long      s_type            /* file system type - cluster size */
};
```



```

/*
 * macros to give more meaningful names to dinfo fields
 */
#define s_m      s_dinfo[0]      /* modulo factor in superblock */
#define s_n      s_dinfo[1]      /* cylinder size in superblock */
#define s_bsize  s_dinfo[2]      /* block size for this file system */

```

If the latter superblock structure is compiled into a program, the native compiler adds pad bytes to force long type data to be aligned on an address that is a multiple of four. A program that attempts to manipulate the PC/IX format superblock must redeclare the values in a manner that does not change the given alignment and then change data references appropriately.

The parameters **NICFREE** and **NICINOD**, the number of in-core free blocks and free i-nodes, respectively, are defined in the system include file, `<sys/param.h>`, as are **BSIZE** (the number of bytes in a block), and **DIRSIZ** (the number of bytes in a simple file name).

The **s_isize** field is the number of the first data block after the i-list; the starts just after the superblock (in block 2); thus the i-list is **s_isize** minus 2 blocks long. The **s_fsize** field is the total number of blocks in the file system. These numbers are used by the system to check for bad block numbers; if a block number that cannot exist is allocated from the free list or is freed, a message is sent to the system console. Moreover, the free array is cleared, to prevent further allocation from a presumably corrupted free list.

The **s_bsize** field contains the number of bytes in a file system block.

The **s_cyl** and **s_skip** fields contain parameters that control the organization of the free-block list. The **s_cyl** field contains the number of blocks per cylinder; **s_skip** is the interleave factor. Free-list interleaving is described by the **mkfs** application. In the PC/IX format file system, these fields are referenced using macros called **s_n** and **s_m**, respectively.

The **s_nicfree** and **s_nicino** fields contain the values of **NICFREE** and **NICINO** (sizes of the **s_free** and **s_inode** arrays). The **s_sicfree** and **s_sincino** fields contain the byte offset from the start of the superblock of **s_free** and **s_inode** arrays. These numbers are provided to facilitate the writing of **BSIZE** independent file system management utilities. These fields are present only in the native format file system.

The free list for each volume is maintained as follows. The **s_free** array contains, in **s_free[1]**, . . . , **s_free[s_nfree-1]**, the block numbers of up to **NICFREE-1** free-blocks. The **s_free[0]** value is the block number of the head of chain of blocks constituting the free list. The first long in each free-chain block is the number (up to **NICFREE**) of free-block numbers listed in the next **NICFREE** longs of this chain member. The first of these block numbers is the link to the next member of the chain. To allocate a block: decrement **s_nfree**, and the new block is **s_free[s_nfree]**. If the new block number is 0, there are no blocks left. This an error condition. If **s_nfree** became 0, read the block named by the new block number, replace **s_nfree** by its first word, and copy the block numbers in the

next **NICFREE** longs into the **s-free** array. To free a block, check whether **s-nfree** is **NICFREE**; if so, copy **s-nfree** and the **s-free** array into it, write it out, and set **s-nfree** to 0. In any event, set **s-free[s-nfree]** to the freed block's number and increment **s-nfree**.

The value of **s-tfree** is the total free-blocks available in the file system.

The value **s-ninode** is the number of free i-numbers in the **s-inode** array. To allocate an i-node: if **s-ninode** is greater than 0, decrement it and return **s-inode[s-ninode]**. If it was 0, read the i-list and place the numbers of up to **NICINOD** free i-nodes into the **s-inode** array, then try again. To free an i-node, if **s-ninode** is less than **NICINOD**, place its number into **[s-ninode]** and increment **s-ninode**. If **s-ninode** is already **NICINOD**, do not bother to enter the freed i-node into any table. This list of in-nodes serves only to speed up the allocation process. The i-node itself indicates whether it is free.

The value of **s-tinode** is the total number of free i-nodes available in the file system.

The **s-fmod** field is a flag to indicate the cleanliness of the file system. A value of 0 indicates that the file system has been cleanly unmounted. Whenever a file system is mounted, this flag is checked and a warning message is printed if the **s-fmod** flag is nonzero. When a clean file system is mounted, the **s-fmod** flag is changed to a value of 1. When an unclean file system is mounted, **s-fmod** is set to 2. When a file system is unmounted, the **s-fmod** flag is reset to 0 only if it has the value 1. Thus, a file system whose **s-fmod** flag is 0 is very likely to be clean, and a file system whose **s-fmod** flag is 2 is likely to have problems.

The **s-only** field is a flag indicating that the file system has been mounted read only. This flag is maintained in memory only; its value on disk is not valid.

The value of **s-time** is the last time the superblock of the file system was changed, (in seconds since 00:00 Jan. 1, 1970 (GMT)).

s-fname is the name of the file system and **s-fpack** is the name of the device on which it resides.

The **s-flock** and **s-ilock** flags are maintained in the copy of the file system in memory while it is mounted; their values on disk are not valid.

The **s-fill**, **s-swaplo**, and **s-nswap** fields are not used on this system.

I-numbers begin at 1, and the storage for i-node 1 begins in the first byte of block 2. I-node 1 is reserved for a file without a name. This i-node is used by the **mkfs** application to put the numbers of defective blocks (blocks with physical flaws) to prevent them from being allocated to other files. I-node 2 is reserved for the root directory of the file system. No other i-number has a built-in meaning. I-nodes are 64 bytes long, so **BSIZE** divide by 64 of them fit into a block. Each i-node represents one file. For the format of an i-node and its flags, see "inode" on page 3-119.

Files

`/usr/include/sys/filsys.h`
`/usr/include/sys/stat.h`

Related Information

In this book: “inode” on page 3-119 and “param.h” on page 4-72.

The **fsck**, **fsdb**, and **mkfs** programs in *AIX Operating System Commands Reference*.

Purpose

Specifies formatting within text files.

Description

A text file format specification normally occurs in the first line of a text file. This format specifies how tabs expand in the remainder of the file.

A format specification consists of a sequence of parameters separated by blanks and enclosed by the brackets `<:` and `>:`. Each parameter consists of a key letter, possibly followed immediately by a value. The following parameters are recognized:

- | | |
|----------------|---|
| d | The d parameter takes no additional value. It indicates that the line containing the format specification is to be deleted from the converted file. |
| e | The e parameter takes no additional value. It indicates that the current format prevails until another format specification is encountered in the file. |
| mmargin | The m parameter specifies a number of spaces added to the beginning of each line. The value of <i>margin</i> must be an integer. |
| ssize | The s parameter specifies a maximum line size. The value of <i>size</i> must be an integer. Size checking is performed after tabs are expanded, but before inserting the margin. |
| ttabs | The t parameter specifies the tab settings for the file. The value of <i>tabs</i> must be one of the following: <ul style="list-style-type: none">• A list of column numbers separated by commas, indicating tabs set at the specified columns.• A <code>-</code> (dash) followed immediately by an integer <i>n</i>, indicating tabs at intervals of <i>n</i> columns.• A <code>-</code> (dash) followed by the name of a supplied tab specification. |

Standard tabs are specified by **t-8**, or the equivalent **t1, 9, 17, 25**, and so on. The **tabs** command defines the supplied tabs.

Default values assumed for parameters not supplied are **t-8** and **m0**. If the **s** parameter is not specified, no size checking is performed. If the first line of a file contains no format specification, the previous defaults are assumed for the entire file.

The format specification can be entered as a comment. In that case it is not necessary to code the **d** parameter.

Example

The following is an example of a line containing a format specification:

```
* <:t5,10,15 s72:> *
```

Related Information

The **ed**, **newform**, and **tabs** commands in *AIX Operating System Commands Reference*.

Purpose

Used as the format for storing graphics file data as graphic primitive strings.

Description

A **GPS** is a graphic primitive string that is used to store graphical data in a particular format. The **plot** and **vtoc** commands produce GPS output files. Several commands edit and display GPS files on various devices. A GPS is composed of as many as five types of graphical data or primitives:

- | | |
|-----------------|---|
| <i>comment</i> | A <i>comment</i> is an integer string included within a GPS file that does not cause anything to be displayed. All GPS files begin with a <i>comment</i> of zero length. |
| <i>lines</i> | A <i>lines</i> primitive has a variable number of points from which zero or more connected line segments are produced. The first point given produces a <i>move</i> to that location, relocating the graphics cursor without drawing. Successive points produce line segments from the previous point. |
| <i>arc</i> | An <i>arc</i> primitive has a variable number of points to which a curve is fit. The first point produces a move to that point. If only two points are given, a line connecting the points is the result. If three points are given, a circular arc through the points is drawn. If more than three points are given, splines are fitted to connect the points. |
| <i>text</i> | The <i>text</i> primitive draws characters beginning at a given point, with the first character centered on that point. |
| <i>hardware</i> | The <i>hardware</i> primitive draws hardware characters or gives control commands to a hardware device. A single point locates the beginning location of the <i>hardware</i> string. |

Graphic primitive strings are given as 16-bit units called **command words**. The first command word determines the primitive type and sets the length of the string. Subsequent command words contain information in multiples of 4 bits of data. The following are the types of GPS and their parameters:

- | | |
|----------------|---|
| <i>comment</i> | <i>cw</i> [<i>string</i>] |
| | <i>cw</i> is the control word. The first 4 bits identify the <i>comment</i> primitive and have the value 0xF. The following bits give the command word count for the primitive. |

[*string*] is a string of characters terminated by a null character. If the string does not end on a command word boundary, another null character is added to align the string with the command word boundary.

lines

cw points sw

cw is the control word. The first 4 bits identify the *lines* primitive and have the value 0x0. The remaining bits give the command word count for the primitive.

points is one or more pairs of integer coordinates having values within a Cartesian plane or universe of 65,536 points on each axis (-32,767 to +32,768).

sw is the style command word. The first 8 bits hold an integer value for *color* information. The next 4 bits contain an integer value for *weight* to indicate line thickness:

- 0 Narrow
- 1 Bold
- 2 Medium.

The last 4 bits of *sw* specify an integer value giving line *style* information:

- 0 Solid
- 1 Dotted
- 2 Dot-dashed
- 3 Dashed
- 4 Long dashed.

arc

cw points sw

cw is the control word. The first 4 bits identify the *arc* primitive and have the value 0x3. The next 12 bits contain the command word count for the primitive.

points is one or more pairs of integer coordinates having values within a Cartesian plane or universe of 65,536 points on each axis (-32,767 to +32,768).

sw is the style command word. The first 8 bits are an integer value for *color*. The next 4 bits contain an integer value for *weight* to indicate line thickness:

- 0 Narrow
- 1 Bold
- 2 Medium.

The last 4 bits contain an integer value setting line style:

- 0 Solid
- 1 Dotted
- 2 Dot-dashed

- 3 Dashed
- 4 Long dashed.

text *cw point fw so [string]*

cw is the control word. The first 4 bits identify the *text* primitive and have the value 0x2. The remaining 12 bits contain the command word count for the primitive.

point is a pair of integer coordinates that are a value within a Cartesian plane or universe of 65,536 points per axis (-32,767 to +32,768).

fw is a font command word. The first 8 bits contain an integer value for *color* information. The next 8 bits contain an integer value for *font* information, with 4 bits giving the *weight* (density) value for the font, and 4 bits giving the *style* (typeface) value for the font.

so is a size/orientation command word. Eight bits specify *textsize* as an integer value to indicate the size of characters drawn. *textsize* represents character height in absolute *universe units*. The actual character height is five times the *textsize* value. The next 8 bits are a signed integer value for *textangle*, and express the angle and direction of rotation of the character string around the beginning *point*. *textangle* is expressed in degrees from the positive *x* axis. The *textangle* value is 256/360 of its absolute value.

hardware *cw point [string]*

cw is the control word. The first 4 bits identify the *hardware* primitive and have the value 0x4. The next 12 bits indicate the command word count for the primitive.

point is a pair of integer coordinates that are values within a Cartesian plane or universe of 65,536 points on each axis (-32,767 to +32,768). This *point* is the starting point for the *string*, which is a string of hardware characters or control commands to a hardware device.

Related Information

In this book: “stat.h” on page 4-73.

The **stat** and **toc** commands in *AIX Operating System Commands Reference*.

group

Purpose

Identifies a group and contains group descriptions.

Synopsis

```
#include <grp.h>
```

Description

Users can be assigned to one or more groups, each of which shares certain protection privileges. The person who sets up the system may want to place users in the same group because they need access to a common set of files. Similarly, a certain group of users can have access only to certain files.

When users log in, they are assigned to the group specified in the **passwd** file. In addition, they are assigned as a member of all groups specified in this file. Users have access to any files that the group to which they are assigned has access. However, any files the user creates can be accessed only by the members of the primary group of which that user is a member. A user can change his primary group for the duration of the terminal session using the **newgrp** command.

The **group** file defines group membership. Each line in this file defines a group and consists of four fields separated by colons.

The format of the data in **/etc/group** is

```
group : ! : gid : user,user, ...
```

These fields are:

<i>group name</i>	A character string of up to 8 characters that references the group.
<i>password</i>	If this field contains a ! (exclamation point). It is placeholder indicating that the encrypted password is stored in the /etc/security/group file.
<i>group ID</i>	A number assigned to the group and used in access decisions.
<i>user group list</i>	A list that specifies the login names of all users in the group. User IDs in the list are separated by commas. The user group list may contain up to 500 eight-character login names.

New systems have at least two groups: the staff group and the system group. New users and groups can be added as necessary.

group

If several users wish to share the same privileges, including the ability to end each other's processes as well as to access the files of others, the same numerical user ID is assigned to each. This ability is also used to give the same person several accounts on the system, each with potentially different login directories and other characteristics, such as electronic mailboxes or login programs.

The **group** file format is a pair of files, **/etc/group** and **/etc/security/group**. These files contain all the information that defines a group of users on the system. The **users** command maintains these files. Programs should use the **getgrent** subroutines to extract fields from the files, and the **putgr** subroutine to update fields for a user.

group The name of the group. This name must also be the name of a stanza in **/etc/security/group**; if such a stanza does not exist, the **sysck** command will create the stanza.

gid The group ID. This ID must be a decimal number

users The members of this group. When a member of the group logs in to the system, this group will be added to the user's concurrent group list.

Specifying the group password allows users who are not normally group members to get temporary group access. Only the **newgrp** command uses the group password.

Example

The following is an example of a group file. It is an ASCII file in which each group is separated from the next by a new-line character. The fields are separated by colons and the file resides in **/etc/group**.

```
system!:0:su,bill,jack,gary
staff!:1:
bin!:2:su,bin
sys!:3:su,bin.sys
adm!:4:su,bin.adm
mail!:6:su
usr!:100:guest
```

The encrypted password (represented by ! (exclamation point)) for a group is stored as an attribute in the file **/etc/security/group**. Only the superuser has access to this file.

The format of the data in **/etc/security/group** is:

```
group:
    password = <encrypted string>
```

Files

/etc/group

/etc/security/group

Related Information

In this book: “passwd” on page 3-167, “a64l, l64a” on page 2-13, “crypt, encrypt” on page 2-116, “getpwent, getpwuid, getpwnam, setpwent, endpwent” on page 2-376, “putpw, putpwent, putpwsent” on page 2-579, and “ulimit” on page 2-834,

The **login**, **newgrp**, **passwd**, and **users** commands in *AIX Operating System Commands Reference*.

history

history

Purpose

Contains the history of an installed licensed program product.

Description

Each licensed program or component of a licensed program that is shipped by IBM contains a history file. The purpose of a history file is to identify the installed release and version of a licensed program or component and to provide a record of any updates (level changes). A history file is replaced when a component is reinstalled. History files are contained on two AIX minidisks, **/usr** and **/vrm**, in the following directories: **/usr/lpp/pgm-name/lpp.hist** and **/vrm/lpp/pgm-name/lpp.hist**, where *pgm-name* is the name of the licensed program or component.

The history file consists of a series of 80-character records. The first two records contain the install data and all subsequent records contain update data. There are three different formats of 80-character records:

Record	Description
Information	Identified by an a , c , r , or v character in position 1. The install and update procedures use information records to identify the licensed program or component name; the current version, release, and level; the date the record was added; and the user who initiated the install or update. Figure 3-4 on page 3-117 shows the format of the fields in the information records.
Title	Identified by a t in character position 1. Contains the descriptive title (up to 30 characters) for the licensed program or component, starting in character position 3. The title record must always be the second record in the history file.
Comment	Identified by an * (asterisk) in character position 1. Allows descriptive comments to be entered into the history file. An * is usually placed in character position 79 to ensure a full 80-column record.

The last character of each record (character position 80) must be a new-line character. Unused character positions must be blank-filled. Tab characters are not permitted.

The first record in a history file must be an information record with a **c** in character position 1. The second record must be the title record. These two records contain data about the installation of the program. The remaining records in the file can be any

combination of information and comment records, and they identify updates to the program.

Figure 3-4 shows the format of an information record in the history file. The definitions for each of the fields other than character position 1 are explained following the figure.

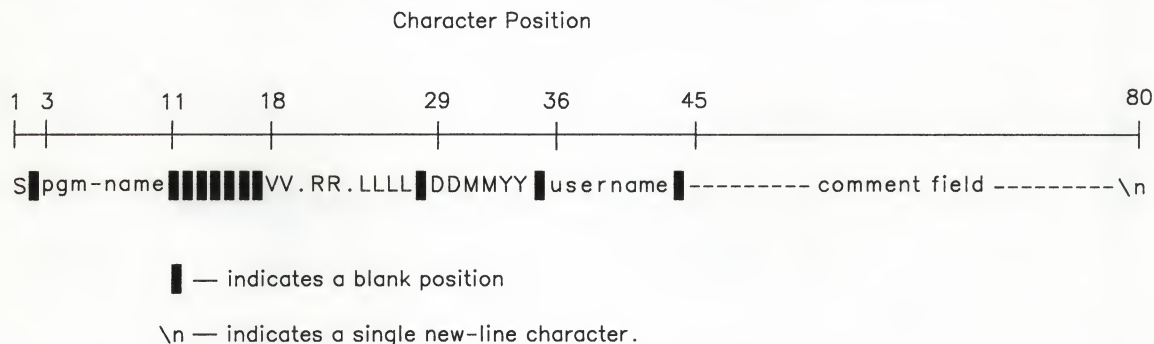


Figure 3-4. Information Record Format

Field	Description
<i>S</i>	The type of information record: <ul style="list-style-type: none"> a Indicates that the update has been applied. c Indicates that the update or install has been committed (accepted). r Indicates that the update has been rejected. v Indicates that the VRM minidisk has been modified.
<i>pgm-name</i>	The name assigned to the program (lowercase characters only). If the name contains less than 8 characters, it must be padded with blanks.
<i>VV.</i>	A 2-digit numeric field followed by a period indicating the version number of the program. The version number indicates the level of the hardware and operating system with which the program works.
<i>RR.</i>	A 2-digit numeric field followed by a period indicating the release number of the program. The release number tracks changes to external programming interfaces since the last version change. This number is generally incremented each time the external interface to the program changes.
<i>LLLL</i>	A 4-digit numeric field indicating the update level of the program. This field is incremented when the changes to the program do not affect external programs that may use the documented external interface for the program. The level, together with the <i>S</i> field, ensures that all changes up to and including the current change are installed on the system.

history

	The fourth (or units) digit of the level is normally 0. IBM reserves this digit for future use.
<i>DDMMYY</i>	These three numeric fields indicate the date the program changed: <i>DD</i> Day of the month (1 to 31). <i>MM</i> Month of the year (1 to 12). <i>YY</i> Year (00 to 99).
<i>username</i>	An alphanumeric field that contains the user name of the person who installed the program. If the user name is shorter than 8 characters, it must be padded with blanks.
<i>comment field</i>	A 35-character field for comments. An * (asterisk) is usually placed in character position 79 to ensure a full 80-column record.
<i>\n</i>	A required new-line character, indicating the end of the record.

Files

/usr/lpp/pgm-name/lpp.hist
/var/lpp/pgm-name/lpp.hist

Related Information

The **installp** and **updatep** commands in *AIX Operating System Commands Reference*.

inode

Purpose

Describes a file system file or directory entry as it appears on a disk.

Synopsis

```
#include <sys/types.h>
#include <sys/ino.h>
```

Description

An **inode** for an ordinary file or directory in a file system has the following structure defined by **sys/ino.h**:

```
/* Inode structure as it appears on a disk block. */
struct dinode
{
    ushort di_mode;      /* mode and type of file */
    short di_nlink;      /* number of links to file */
    ushort di_uid;       /* owner's user id */
    ushort di_gid;       /* owner's group id */
    off_t di_size;        /* number of bytes in file */
    char di_addr[40];     /* disk block addresses */
    time_t di_atime;      /* time last accessed */
    time_t di_mtime;      /* time last modified */
    time_t di_ctime;      /* time created */
};
/*
 *the 40 address bytes:
 * 39 used; 13 addresses
 * of 3 bytes each.
 */
```


inode

The fields in the structure are as follows:

addr	Array of thirteen 3-byte block numbers assigned to this file. The first 10 block numbers are direct addresses while the last 3 are indirect addresses.
atime	Time this file was last accessed.
ctime	Time this file was created.
gid	Group ID.
mode	Type and access permissions of file.
mtime	Time this file was last modified.
nlink	Number of directories that name this file.
size	Number of bytes in file.
uid	Owner ID.

See the **types** file for related information concerning the **off_t** and **time_t** define types.

Related Information

In this book: “fs” on page 3-100, “stat.h” on page 4-73, and “types.h” on page 4-78.

kaf

Purpose

Specifies how to process **ddi** keywords and their parameters.

Description

Keyword Attribute files, also called **kaf** files, define how the **devices** command and customize helpers are to process keywords used in **ddi** files. The **kaf** files:

- Contain instructions for processing device information
- Control whether the **devices** command displays the associated information
- Control whether a user can change the information using the **devices** command
- Specify the input validation that the **devices** command performs
- Determine the action that the customize helper takes.

The **kaf** information can be included in the **ddi** file for the device, or it can appear in a separate file. If it is contained in a separate file, then the stanza for the device in the **system** file must name the **kaf** file as the value of the **kaf_file** keyword. The **kaf_use** keyword (also in the **system** file) specifies the stanza of the **kaf** file to use.

The name of each stanza in a **kaf** file is the name of a keyword that is used in **ddi** files. The stanza defines how the **devices** command and customize helper programs process that **ddi** keyword. The following section defines the keywords that can appear in the stanzas of **kaf** files.

The use of extended characters in **kaf** files is not supported.

Directives to the Customize Helper

add	Specifies the actions for the customize helper to take during a vrconfig -a (add) operation.
delete	Specifies the actions for the customize helper to take during a vrconfig -d (delete) operation.
startup	Specifies the actions for the customize helper to take during a vrconfig -startup operation.
shutdown	Specifies the actions for the customize helper to take during a vrconfig -shutdown operation.

The value for each of the preceding keywords has the format *x/y*, where *x* and *y* can be any of the following:

- b** Constructs the **Define-Device** structure, including the Block I/O Communication Area (BIOCA) device characteristics. Appends other device characteristics specified by the programmer. (See the discussion of customizing helpers in *AIX Operating System Programming Tools and Interfaces*.) Performs the **Define-Device** SVC by issuing an **ioctl** system call to the **config** device driver.
- n** Constructs the **Define-Device** structure. Appends other device characteristics specified by the programmer. (See the discussion of customize helpers in *AIX Operating System Programming Tools and Interfaces*.) Performs the **Define-Device** SVC by issuing an **ioctl** system call to the **config** device driver.
- u** Constructs the kernel device driver structure and issues an **ioctl** system call to the **config** device driver to initialize the driver.

Action to Take after Customization

- syschg** Specifies the action the **devices** command takes when the user changes a device characteristic. The valid choices are:
- a** Rebuilds the kernel and IPLs the system
 - s** Runs the special processing routine specified by the **specproc** keyword in the **/etc/system** stanza.
 - none** Takes no special action.

Control over Display and Modification of the Keyword

- display** If set to **TRUE**, then the **devices** command displays the device characteristic keyword and allows the user to change its value.
- dsrc** Determines whether the **devices** command displays the adapter characteristic keyword from the **ddi** file. The value is a list of adapter numbers, separated by commas. The **devices** command displays the keyword and allows the user to change its value only if the number of the adapter associated with the device appears in the list.
- required** If set to **TRUE**, the **devices** command displays the keyword and advises the user to make sure that its value matches the system configuration. **devices** does not check to see whether the entered value matches the system configuration.
- rsrc** Determines whether the **devices** command displays the adapter characteristic keyword from the **ddi** file and requires the user to enter a value. The value is a list of adapter numbers, separated by commas. If the number of the adapter associated with the device appears in the list, then the **devices** command displays the keyword and requires the user to specify its

value. **devices** does not check to see whether the entered value matches the system configuration.

User Input Validation

vtype Specifies the type of checking that the **devices** command performs on values entered by the user. **vtype** can be set to one of the following values:

- 0 No validation.
- 1 Mapping validation: the input value must be one of the keywords found in the stanza named by the **map** keyword.
- 3 Range validation: The input value must have the data type specified by the **type** keyword and must fall in the range specified by the **range** keyword.

map Names a stanza in the **kaf** file that contains a list of *keyword=value* pairs against which the input value is to be matched. If the input matches a given *keyword*, then the corresponding **value** is substituted in its place.

opts Specifies the *options* to search for in the **/etc/ddi/options** file. **opts** is one of the following:

- k** Keyword only
- a** Keyword followed by adapter name
- c** Keyword followed by device class
- t** Keyword followed by device type
- s** Keyword followed by device stanza name.

If the **opts** keyword is not specified, its value defaults to **k**. See "options" on page 3-165 for details about the **/etc/ddi/options** file.

range Defines the valid range of values for a keyword so that the **devices** command can verify values entered by the user. The value of the **range** keyword has the format *first,last,incr*, where *first* is the first number in the range, *last* is the last number, and *incr* is the increment between values in the range. For example, **range=2,10,2** specifies the values 2, 4, 6, 8, and 10.

type Defines the data type for the value of a keyword. The **devices** command ensures that the values entered are the correct data type, specified by one of the following:

- F** Floating-point (**float**)
- H** Hexadecimal (**int**)
- I** Integer (**int**)
- L** Long integer (**long int**)
- S** Short integer (**short int**)
- U** Unsigned integer (**unsigned int**).

Files

`/etc/ddi/font.kaf`
`/etc/ddi/opprinter.kf`
`/etc/ddi/osprinter.kf`
`/etc/ddi/plotter.kaf`
`/etc/ddi/pprinter.kaf`
`/etc/ddi/sprinter.kaf`
`/etc/ddi/tty.kaf`
`/etc/mdkaf`

Related Information

In this book: “attributes” on page 3-20 “ddi” on page 3-47, “descriptions” on page 3-65, “system” on page 3-210, and “config” on page 5-21.

The discussion of customize helpers in *AIX Operating System Programming Tools and Interfaces*.

master

Purpose

Contains master configuration information.

Description

The **master** file is an attribute file that contains stanzas that describe all device drivers defined in the system. There are two kinds of stanzas: kernel device driver stanzas and VRM driver stanzas. The kernel driver stanzas specify drivers to link into the kernel and the VRM drivers that support them. VRM driver stanzas specify drivers to be loaded into the VRM at the time the system is loaded.

The use of extended characters in the **master** file is not supported.

The sysparms Stanza

The first stanza of the **/etc/master** file, the **sysparms** stanza, defines the values for many system parameters and limits. If you need to modify any of these system parameters, first make the changes in the **/etc/master** file, then rebuild the kernel. See *Device Driver Development Guide* for instructions on rebuilding the kernel.

callouts Specifies the number of callouts the kernel uses for event waiting.

charlists Specifies the number of character lists the terminal driver uses.

connections Specifies the maximum number of concurrent network connections allowed for Distributed Services.

Note: This keyword sets the size of the node table in the kernel.

drivernkprocs Specifies the maximum number of kernel processes available for a given device driver. To create this kind of keyword, replace *driver* with an abbreviation for the device driver, then follow it with the letters **nkprocs**. (For example, **tcpipnkprocs**.)

Note: A keyword ending with the letters **nkprocs** should be defined in the **/etc/master** file for any device driver that needs to allocate kernel processes.

dsnkprocs Specifies the maximum number of kernel-defined processes available for use by Distributed Services.

dumpdev Specifies the target device for kernel dumps.

filetab	Specifies the number of entries in the kernel open file table.
floating	Indicates whether the kernel should attempt to use floating-point hardware, if it is present. Options in this parameter are hardware and software . The default, software , means there is no optional floating-point accelerator hardware.
hashbuffers	Specifies the number of hash buffers the kernel uses.
hftbuffers	Specifies the number of virtual terminals.
inodetab	Specifies the number of entries in the kernel i-node table.
iobuffers	Specifies the number of physical I/O buffers the kernel supports.
kbuffers	Specifies the number of disk buffers in the kernel. If kbuffers is not defined, or if it is set to 0, then the system chooses the number of buffers based on the processor and the amount of real memory installed in the system.
kdmabuffers	Specifies the number of buffer headers used by the exec system call. The default value is 32.
kmap	Specifies the number of elements in resource map array for internal kernel storage.
kprocs	Specifies the number of kernel defined processes the kernel supports.
maxprocs	Specifies the maximum number of processes for each terminal session.
mountab	Specifies the number of entries in the mount table used by the kernel for file system mounting. This value sets the limit of the number of file systems that can be mounted by the kernel. See "mnttab" on page 3-161 for details.
msgmap	Specifies the number of elements in the message map array. This array assists in the allocation of space within the message data area. Elements of the array represent free space in the message data area. This space is created by the return of active messages. The value of the msspmap parameter should be approximately half the msgtql value.
msgmax	Specifies the maximum number of bytes allowed for a single message.
msgqid	Specifies the maximum number of active message queues.
msgqmax	Specifies the maximum number of bytes allowed for a single message queue.

msgseg	Specifies the number of segments in the message data area. This number must be less than 32,768. Note: Multiplying the msgseg by the msgsegsz gives you the size of the data area. Increasing one or both of these values enlarges the message data area. When adjusting the data area, increase the number of segments before increasing the size of segments.
msgsegsz	Specifies the size of each segment in the message data area. The size is in bytes and should be a multiple of four. This is the minimum allocation of space for a message. Regardless of the space required, an entire segment is used. Messages which use less than an allocated segment waste space. A small msgsegsz is most efficient; 8 bytes is an adequate specification.
msgtql	Specifies the maximum number of active messages permitted in the system.
nflocks	Specifies the maximum number of simultaneously locked file regions.
netnoone	Specifies the Distributed Services user ID or group ID value to use when no user ID from the local node maps to a remote file's owner ID. The default value is 0xFFFF.
netsomeone	Specifies the Distributed Services user ID or group ID value to use when more than one local ID maps to a remote file's owner ID, and one particular ID cannot be selected (for example, because of wildcard mappings). The default value is 0xFFFE.
nid	Specifies the node ID to generate into the system. This keyword is currently unused.
nncb	Specifies the size of the Distributed Services translate table array. Each node for which the kernel has user ID or group ID translate information has an entry in this array of translate headers.
node	Specifies the node name to generate into the system.
pinkbuffers	Specifies whether or not the disk buffers should be pinned. The value can be true or false .
pipdev	Names the stanza that defines the file system used for FIFO files.
power	Indicates whether the kernel has power warning code. If this value is true , power warning code exists. The default is false .
procs	Specifies the total number of simultaneous processes the kernel supports.
pslotkill	Specifies the threshold at which the system begins to kill processes in order to recover paging space. pslotkill is specified in slots, where a slot is 2048 bytes (four blocks) of a paging minidisk. The default value is 200 slots.

pslotpanic	Specifies the threshold at which to stop AIX and attempt a system dump because paging space has almost been exhausted. Note that the system dump itself cannot finish because of the lack of paging space. pslotpanic is specified in slots, where a slot is 2048 bytes (four blocks) of a paging minidisk. The default value is 100 slots.
pslotwarn	<p>Specifies the threshold at which the system displays a message warning that paging space is running low. When the system displays this message, it also:</p> <ul style="list-style-type: none">• Performs a sync to write all changes to disk• Enters sync mode, in which disk I/O is not buffered• Sends all processes the SIGDANGER signal to warn them that the system is likely to crash any moment. <p>pslotwarn is specified in slots, where a slot is 2048 bytes (four blocks) of a paging minidisk. The default value is 350 slots.</p>
ptybuffers	Specifies the number of pseudo-terminals that can be present in the system (see “pty” on page 5-126). The maximum value for ptybuffers is 256, while the default value is 16.
release	Specifies the operating system release number to generate into the system.
rootdev	Names the stanza in the /etc/system file that defines the root file system device.
rsbuffers	Specifies the number of buffers allocated for the asy terminal driver.
semadjmax	Specifies the maximum value allowed for semaphore adjust value on exit.
semid	Specifies the number of distinct semaphore identifiers the kernel supports.
semmap	Specifies the number of entries in semaphore map array.
semmax	Specifies the maximum number of simultaneous semaphores allowed and supported by the kernel.
semopmax	Specifies the maximum number of operations allowed for each semop system call.
semsetmax	Specifies the maximum number of semaphores allowed in a set.
semunmax	Specifies the number of semaphore undo structures the kernel supports.
semunpmax	Specifies the maximum number of undo entries for each process.
semvalmax	Specifies the maximum value allowed for each semaphore.
shmid	Specifies the number of distinct shared memory identifiers the kernel supports.

shmmax	Specifies the maximum number of kilobytes for shared memory allowed per shared segment.
shmmin	Specifies the minimum number of kilobytes for shared memory allowed per shared segment.
shmsegs	Specifies the number of segment registers that may be used to support shared memory.
shmsysmax	Provides compatability with other UNIX systems. This value is otherwise ignored.
slice	Specifies the percentage of time in <i>quanta</i> that a process can run before it must relinquish control of the processor. Each <i>quantum</i> on the RT system is equal to 333 milliseconds.
system	Specifies the system name to generate into the system.
texttab	Specifies the number of shared text segment entries in the the text table.
version	Specifies the version number of the operating system to generate into the system.

Kernel Driver Stanzas

There is a unique set of keywords associated with each type of stanza. It is not necessary, however, for a stanza to contain all the keywords associated with that type. If a keyword is omitted from the stanza, the default is used. Mandatory keywords must be supplied and are not defaulted. The name of each stanza is a logical kernel driver name referenced in other stanzas.

The lines interpreted by the **config** and **vrnconfig** commands are:

config	Indicates that this device has a customization helper program, which provides assistance in decoding other options. This value is the name of the helper program in the <i>/etc</i> directory. See "config" for more information about customization helper programs.
major	Identifies the major device number for this driver. This is mandatory.
mandatory	Identifies this driver to be included whether or not the system file asks for it. If this value is TRUE , include this driver.
maxminor	States the maximum number of minor devices this driver supports. This number should agree with the driver code.
mpx	Identifies a multiplexed special file when this value is TRUE .
prefix	Provides a prefix for the driver routines. For example, if this value is <i>abc</i> , then the open routine in the driver is <i>abcopen</i> . This keyword is mandatory. Note that all drivers are assumed to be archived into the system object libraries.

master

routines	Identifies the routines actually defined for this driver. The possible routines are open , close , read , write , strategy , ioctl , init , and print .
tty	Identifies whether the device is a terminal. If this value is TRUE , the device is a terminal and terminal structures are defined.
vdriver	Names the VRM driver stanzas for the related VRM drivers.

Other lines can be included for interpretation by customization helper programs.

VRM Driver Stanzas

The **iocn** lines identify VRM driver stanzas. The name of each stanza is a logical VRM driver name referenced in other stanzas.

The lines interpreted by the **vrconfig** command are:

code	Specifies the full path name of the file containing executable VRM code that contains the table of contents format of the VRM driver.
copy	Names a previously specified VRM driver stanza to be used instead of the code keyword specification.
ctype	Indicates the code type, such as vdrrvr . This is an informational keyword for IBM customization helpers.
iocn	Assigns the decimal I/O code number to this driver.
protocol	If the value is TRUE , indicates that this stanza describes a protocol procedure.

Other lines can be included for interpretation by customizaton helper programs.

Miscellaneous System Parameters

Both the **master** and the **system** file can have option lines describing miscellaneous system customizing and tuning options in the **sysparms** stanzas. Options in the **system** file override those in the **master** file. These options include:

inetlen	Specifies the Internet packet length for file transfer. (See the xftp command in <i>Interface Program for use with TCP/IP</i> .) The default value is 1064 bytes.
level	Specifies the level number of the operating system to generate into the system.
msgheader	Specifies the maximum number of system message headers allowed.

Other keywords can be added as needed.

Example

The following sample of a master file entry contains AIX Operating System and VRM information.

- * Kernel drivers, identified by "major" keyword

- * printer drivers

u5182mp:

```
major = 6
prefix = lp
routines = open,close,write,ioctl,init
maxminor = 8
vdriver = v5182mp
config = vrcmain
```

u5182sp1:

```
major = 6
prefix = lp
routines = open,close,write,ioctl,init
maxminor = 8
vdriver = v5182sp1
config = vrcmain
```

u5182sp2:

```
major = 6
prefix = lp
routines = open,close,write,ioctl,init
maxminor = 8
vdriver = v5182sp2
config = vrcmain
```


master

* VRM driver entries

v5182mp:

iocn = 2014
code = /vrml/vrmdd/vpptr
ctype = vdrv

v5182sp1:

iocn = 2015
code = /vrml/vrmdd/vpptr
ctype = vdrv

v5182sp2:

iocn = 2016
code = /vrml/vrmdd/vpptr
ctype = vdrv

File

`/etc/master`

Related Information

In this book: “mount” on page 2-460, “vmount” on page 2-861, “mnttab” on page 3-161, “attributes” on page 3-20, “system” on page 3-210, and “pty” on page 5-126.

The **vrmlconfig** and **config** commands in *AIX Operating System Commands Reference*.

```

        /* flag definitions for MSGHEADR */
#define mih_status 0x0001    /* off (0): status = null */
                             /* on (1): status = current */

```

Each entry in the insert index table contains only the header information.

```

struct ins_indx {           /* Insert table entry */
                             /* (contains header info. only) */
    MSGHEADR                /* header information */
};

```

Each entry in the message index table and help index table contains the header information plus the title length (used for helps), a message/help manual reference, and the index number for the help associated with a message.

```

struct mih_indx { /* Entry in message or help index tables. May */
                  /* also be used as entry in insert index table */
                  /* if only header information is referenced. */
    MSGHEADER      /* header information */
    unsigned short titlelen; /* title length (not incl null term) */
    char           dcompid[3]; /* displayed component ID */
    char           dmsgid[3];  /* displayed message help ID */
    short          helpindx;   /* help index number (zero if no help, */
                              /* negative if help in common file) */
    unsigned short reserved;   /* reserved (zero) */
};

```

Each index table must be aligned on a long integer boundary.

Related Information

In this book: “msghelp” on page 2-471, “msgimed” on page 2-474, “msgqued” on page 2-478, and “msgtrtv” on page 2-486.

The **gettext** and **puttext** commands in *AIX Operating System Commands Reference*.

Mfile

Mfile

Purpose

Contains the log of medium access control (MAC) frames.

Description

When Token-Ring diagnostics are started with the **-M** flag, MAC frames that are received and analyzed are logged. The **/etc/trdconf** file specifies the default file where MAC frames are logged. The **/etc/trdconf** file provided with the system contains the default MAC frame log file name of **/usr/adm/ras/Mfile**. Data contained in the MAC frame log file is in binary form.

Each MAC frame record in the log file is preceded by a header that has the same format as the **errhdr** structure defined for the error log in the **sys/erec.h** file. The structure definition for this header has the following format:

```
struct errhdr {
    unsigned e_len;      /* bytes in record (with header) */
    time_t   e_time;     /* time of day */
    long     e_timex;    /* clock ticks */
    char     e_nid[8];   /* Node ID */
    char     e_vmid[8];  /* Virtual Machine ID */
    union {
        struct {
            char ex_class;
            char ex_subclass[2];
            char ex_type; /* record type */
        } ex;
        int csmt;
    } exx;
};
```

The fields in the structure above are used in the following manner by the MAC frame log file:

e_len	The number of bytes in the record (including the header).
e_time	The time in seconds as returned by the time system routine.

e_timex	This field is not used and is set to 0.
e_nid	The node ID of the machine.
e_vmid	This field is not used and is set to blanks.
ex_class	Set to E_INFO as defined in erec.h .
ex_type	Set to E_INFO as defined in erec.h .
ex_subclass	The first byte of this field is set to E_APPL and the second byte to E_TRD as defined in erec.h .

The format of the MAC frame data varies with the particular MAC frame. The following MAC frames are received and processed by the Token-Ring diagnostics:

- Beacon, X '02'
- Report Neighbor Notification Incomplete, x '27'
- Report Active Monitor Error, X '28'
- Report Soft Error, X '29'
- Report Ring Station Addresses, X '22'

For a description of the individual MAC frames, see *IBM Token-Ring Network Architecture Reference*.

Files

/usr/adm/ras/Mfile
/etc/trdconf

Related Information

In this book: “trdconf” on page 3-242.

“Token-Ring Diagnostics” in *Managing the AIX Operating System*.

The **trdiag** command in *AIX Operating System Commands Reference*.

Purpose

Defines aliases for the Message Handling (MH) Package.

Description

The Message Handling (MH) Package supports both personal alias files and a system-wide alias file, **/usr/lib/mh/MailAliases**. Depending on the MH configuration, MH may also honor a system-wide alias defined for the **sendmail** command. An alias file contains lines that associate an alias name with an address or a group of addresses.

File Format

Each line of an alias file has one of the following formats:

alias : *address-group*

alias ; *address-group*

< *file*

where

address-group := *address-list*
 | < *file*
 | = *AIX-group*
 | + *AIX-group*
 | *

address-list := *address*
 | *address-list*, *address*

You can continue an alias definition on the next line by ending the line to be continued with the \ (backslash) character followed by the new-line character.

The *alias-file* and *file* parameters must be AIX file names. *AIX-group* must be a group name (or number) from **/etc/group**. *address* must be a simple Internet-style address. MH treats alias file names as case-sensitive names but ignores case elsewhere in alias files.

If a line starts with the < (less than) character, MH reads the file specified after the < for more alias definitions. The reading is done recursively.

If an address group starts with the < (less than) character, MH reads the file specified after the < and adds the contents of that file to the address list for the alias.

message

Purpose

Describes message, insert, and help formats.

Synopsis

```
# include <msg10.h>
```

Description

The **puttext** command is used to convert message, text insert, and help descriptions from a format that can be edited into a format that can be accessed at run time. The descriptions in the file can be accessed by using the **msgimed**, **msgqued**, **msghelp**, and **msgtrtv** subroutines. The **gettext** command converts the descriptions back into a format that can be edited.

The file header contains a unique identifier indicating the type of file, a file format version number (currently 0), and the number of component entries in the file (currently, only one component entry per file is supported). The header file has the following form:

```
struct filehdr {          /* FILE HEADER */
    char        unique[8]; /* unique file identifier "MSGFILE" */
    unsigned short version; /* file format version number */
    unsigned short numcomp; /* number of component entries in file */
};
```

Following the file header is the component index table. Each entry (currently, there is only one) in the table identifies the component, the national language (EN for English), the maximum index numbers that have been allocated and the offsets to the message index table, insert index table and help index table.

message

```
struct cmp_indx {          /* Component index table entry */
    char        compid[6];  /* component ID */
    char        langid[2];  /* language ID */
    unsigned short flags;   /* reserved for flags (zero) */
    unsigned short maxnum[3]; /* max index numbers used for */
                                /* messages, inserts, and helps */
    unsigned long offset[3]; /* offsets to msg, insert, and help */
                                /* index tables from start of file */
    unsigned long reserved; /* reserved */
};
```

The component index table is followed by the message index table and message text, the insert index table and insert text, and help index table and help text. The header for each entry in the message, insert, and help index tables identifies the component ID and index number where the text actually resides, the offset to the text (and its length) if the text actually resides in this entry, the version number (used with a common file), and an indicator of whether the entry is current (can be accessed) or null.

```
/* Format of header for entries in the */
/* message, insert, and help index tables */
/* (Note that each index table must be */
/* aligned on a long integer boundary.) */
```

```
#define MSGHEADR
    char        compid[6];  /* component ID for text source */
                                /* file ('=====' or 'common') */
    unsigned short index;   /* index # for text source (zero */
                                /* indicates same index # */
    unsigned long offset;   /* offset to text from start of */
                                /* index table */
    unsigned short textlen; /* text length (not incl null term) */
    unsigned short version; /* version */
    unsigned short flags;   /* flag definition */
                                /* 01 off: status = null */
                                /*      on: status = current */
                                /*      (other flags reserved (zero)) */
    unsigned short reserve1; /* reserved (zero) */
```

If an address group starts with the = (equal) character, MH consults the **/etc/group** file for the AIX group specified after the equal character. MH adds each login name occurring as a member of the group to the address list for the alias.

If an address group starts with the + (plus) character, MH consults the **/etc/group** file to determine the group ID of the AIX group specified after the plus character. MH adds each login name occurring in the **/etc/passwd** file whose group ID is indicated by this group to the address list for the alias.

If an address group is specified as * (asterisk), MH consults the **/etc/passwd** file and adds all login names with a UID greater than 200, or the value set for everyone in **/usr/lib/mh/mtstailor** to the address list for the alias.

In matching, a trailing * in an alias matches just about anything appropriate. See "Example" on page 3-140.

The following procedure approximates how the system resolves aliases at posting time:

1. The system builds a list of all addresses from the message to be delivered, eliminating duplicate addresses.
2. If the draft originated on the local host, the system performs alias resolution for those addresses in the message that have no specified host.
3. For each line in the alias file, the system compares the alias with all existing addresses. If a match is found, the system removes the matched alias from the address list, and adds each new address in the address group to the address list if it is not already in the list. The alias itself is not usually output; the address group that the alias maps to is output instead. If the alias is terminated with a ; (semicolon) instead of a : (colon), both the alias and the address are output in the correct form. (This correct form makes replies possible since MH aliases and personal aliases are unknown to the mail transport system.)

In the MH system, aliases in alias files are expanded into the headers of messages posted. This aliasing occurs first, at posting time, without the knowledge of the message transport system. In contrast, once the message transport system is given a message to deliver to a list of addresses, for each address that appears to be local, a system-wide alias file is consulted. These aliases are not expanded into the headers of messages delivered.

Since alias files are read line by line, forward references work, but backward references are not recognized. Although this forward-referencing semantics prevents recursion, the *<alias-file>* syntax may defeat this. Since the number of file descriptors is limited, such recursion will end when all file descriptors are depleted.

Example

The following example of a **mh-alias** file illustrates some features:

```
</user/lib/mh/DraftingAlias
temps:peggy,tina
tina:temp5@NODE3
p1:<project1.aliases
staff:=staff
support:+syssup
everyone:*
news.*:news
```

The first line says that more aliases should immediately be read from the file `/usr/lib/mh/DraftingAliases`. Following this, `tina` is defined as an alias for `temp5@NODE3`, and `temps` is defined as an alias for the two names `peggy` and `tina`. The definition of `p1` is given by reading the file `user-mh-directory/project1.aliases`. `staff` is defined as all users who are listed as members of the group `staff` in the `/etc/group` file, and `support` is defined as all users whose group ID in `/etc/passwd` is equivalent to the `syssup` group. Finally, `everyone` is defined as all users with a user ID in `/etc/passwd` greater than 200, and all aliases of the form `news.anything` are defined to be `news`.

Files

<code>/usr/lib/mh/MailAliases</code>	The default system alias file.
<code>/usr/lib/mh/mtstailor</code>	The MH tailor file.

Related Information

In this book: “group” on page 3-113 and “passwd” on page 3-167.

The **ali**, **conflict**, **post**, **send**, **sendmail**, and **whom** commands in *AIX Operating System Commands Reference*.

“Overview of the Message Handling Package” in *Managing the AIX Operating System*.

mh-format

Purpose

Defines message formats for the Message Handling (MH) Package.

Description

Several MH commands use either a format string (similar to a **printf** string) or a format file to format their output. For example, the **scan** command uses a format string to format the scan listing for each message, while the **repl** command uses a format file to format a message reply.

An MH format string is similar to a **printf** string, but uses multiletter escape sequences beginning with the % (percent) character. In addition, the usual C language backslash characters (**\b**, **\f**, **\n**, **\r**, and **\t**) are recognized. To continue a format line to the next line, precede the new-line character with a **** (backslash).

The interpretation model is based on a simple machine with two registers: *num* and *str*. The former contains an integer value; the latter a string value. When an escape is processed, if the escape requires an argument, the system reads the current value of either *num* or *str*, and if the escape returns a value, the system writes either *num* or *str*.

Escapes

Escapes are of three types: **component**, **function**, or **control**. A component escape, specified as **%{name}**, is created for each header found in the message being processed. For example, **%{date}** refers to the **Date:** field of the appropriate message. A component escape is always string-valued.

A control escape is one of:

```
%<escape  
%!  
%>
```

which corresponds to an if-then-else construct. If *escape* has a nonzero value (for integer-valued escapes) or is not empty (for string-valued escapes), then everything up to **%!** or **%>** (whichever comes first) is interpreted; otherwise, processing skips to **%!** or **%>** (whichever comes first) and starts interpreting again.

mh-format

A function escape is specified as `%(name)` and is statically defined. Here is the list:

Escape	Argument	Returns Interpretation
nonzero	integer integer	<i>num</i> has a nonzero value
zero	integer integer	<i>num</i> is zero
eq	integer integer	<i>num</i> == width
ne	integer integer	<i>num</i> != width
gt	integer integer	width > <i>num</i>
null	string integer	<i>str</i> is empty
nonnull	string integer	<i>str</i> is not empty
putstr	string	Display <i>str</i>
putstrf	string	Display <i>str</i> in the specified width, for example: %20(putstrf{ <i>subject</i> })
putnum	integer	Display <i>num</i>
putnumf	integer	Display <i>num</i> in the specified width, for example: %4(putnum(<i>msg</i>))
msg	integer	Message number
cur	integer	Message is current
size	integer	Size of message
strlen	string integer	Length of <i>str</i>
me	string	The user's mailbox
plus	integer	Add width to <i>num</i>
minus	integer	Subtract <i>num</i> from width
charleft	integer	Space left in output buffer
timenow	integer	Seconds from 00:00:00 GMT January 1, 1970

When *str* is a date, these escapes are useful:

Escape	Argument	Returns Interpretation
sec	string integer	Seconds of the minute
min	string integer	Minutes of the day
hour	string integer	Hours of the day (24-hour clock)
mday	string integer	Day of the month
mon	string integer	Month of the year
wday	string integer	Day of the week (Sunday = 0)
year	string integer	Year of the century
yday	string integer	Day of the year
dst	string integer	Daylight savings in effect
zone	string integer	Time zone
sday	string integer	Day of the week known. Values are: 1 Explicit day 0 Implicit (the MH package figured it out) -1 Unknown (the MH package could not figure it out)
clock	string integer	Seconds from 00:00:00 GMT January 1, 1970
rclock	string integer	Seconds prior to current time
month	string string	Month of the year
lmonth	string string	Month of the year (long form)
tzone	string string	Time zone
day	string string	Day of the week
weekday	string string	Day of the week (long)
tws	string string	RFC 822 rendering of the date
pretty	string string	A more user-friendly rendering
nodate	string	Could not be parsed

mh-format

When *str* is an address, these escapes are useful:

Escape	Argument	Returns Interpretation
pers	string string	The personal name of the address
mbox	string string	The local part of the address
host	string string	The domain part of the address
path	string string	The route part of the address
type	string integer	The type of host. Values are: -1 uucp 0 Local 1 Network 2 Unknown
nohost	string integer	No host was present in the address
ingrp	string integer	The address appeared inside a group
gname	string string	The name of the group (present for first address only)
note	string string	Commentary text
proper	string string	RFC 822 rendering of the address
friendly	string string	A more user-friendly rendering
mymbox	string	The address refers to the user's mailbox
formataddr	string	Display <i>str</i> in an address list

mhl.format

The **mhl.format** file is similar to other MH format files, but controls the format of output when **mhl** is the message listing program. Each line of the **mhl.format** file must have one of the following forms:

```
;comment  
:cleartext  
variable [, variable . . .]  
component : [ variable . . . ]
```

A line beginning with a ; (semicolon) contains comments that are ignored. A line beginning with a : (colon) contains text for output. A line that contains a : character only produces a blank output line. A line beginning with *component* defines the format of the specified component. If a variable follows a component, the variable applies only to that component. Lines having other formats define the global environment.

Variables that have integer or string values as arguments must be followed by an = (equal) character and the integer or string value (for example, `overflowoffset=5`). String values must also be enclosed in double quotation characters (for example, `overflowtext="****"`). An argument specified with the suffix `/G` has global scope. An argument specified with the suffix `/L` has local scope.

The entire **mhl.format** file is parsed before output processing begins. Thus, if a variable's global setting is defined in multiple places, the last global definition for that variable describes the real global setting.

The following table lists the **mhl.format** variables and their arguments:

Variable	Argument	Description
width	integer	Set the screen width or component width
length	integer	Set the screen length or component length
offset	integer	Indent <i>component</i> the specified number of columns
overflowtext	string	Output <i>string</i> at the beginning of each overflow line
overflowoffset	integer	Indent overflow lines the specified number of columns
compwidth	integer	Indent component text the specified number of columns after the first line of output
uppercase	flag	Output text of <i>component</i> in all uppercase characters
nouppercase	flag	Output text of <i>component</i> in the case entered
clearscreen	flag/G	Clear the screen before each page
noclearscreen	flag/G	Do not clear the screen before each page
bell	flag/G	Produce an audible indicator at the end of each page
nobell	flag/G	Do not produce an audible indicator at the end of each page
component	string/L	Use <i>string</i> as the name for the specified <i>component</i> instead of the string <i>component</i>
nocomponent	flag	Do not output the string <i>component</i> for the specified <i>component</i>

Variable	Argument	Description
center	flag	Center <i>component</i> on line. This variable works for one-line components only.
nocenter	flag	Do not center <i>component</i>
leftadjust	flag	Strip off the leading whitespace characters from each line of text
noleftadjust	flag	Do not strip off the leading whitespace characters from each line of text
compress	flag	Change newline characters in text to space characters
nocompress	flag	Do not change newline characters in text to space characters
formatfield	string	Use <i>string</i> as the format string for the specified <i>component</i>
addrfield	flag	The specified <i>component</i> contains addresses
datefield	flag	The specified <i>component</i> contains dates.
ignore	unquoted string	Do not output component specified by <i>string</i>

Example

The following format string is the default for the **scan** command. It has been divided into several pieces for readability. The first part is:

```
%4(putnum(msg))%<(cur)+%i %>%<{replied}-%i %>
```

This says that the message number should be displayed in four digits. If the message is the current message, a + character is displayed next; otherwise, a space character is displayed. If a **Replied:** field is present, then a - (minus) is displayed; otherwise, a space is displayed. Next:

```
%02(putnumf(mon{date}))%02(putnumf(mday{date}))
```

The month and day are displayed in two digits (zero filled). Next:

```
%<{date} %i*>
```

If no **Date:** field is present, then an * (asterisk) character is displayed; otherwise, a space character is displayed. Next:

```
%<(myinbox{from})To:%14(putstr(friendly{to}))
```


If the message is from me, display **To:** followed by a friendly rendering of the first address in the **To:** field. Continuing:

```
%!%17(putstrf(friendly{from}))%
```

If the message is not from me, display **From:** followed by the **From:** address. And finally:

```
%{subject}<<{%body}>>
```

Display the subject and initial body of the message.

This method of formatting messages allows you to extract individual fields and display them in the format you desire.

If you use the **-form file** argument when you run **scan**, it treats each line in *file* as a format string and acts accordingly. The following files contain **scan** listing formats that you can look at: **/usr/lib/mh/scan.time**, **/usr/lib/mh/scan.size**, and **/usr/lib/mh/scan.timely**.

The following line is an example of a line that could appear in the **mhl.format** file:

```
width=80,length=40,clearscreen,overflowtext="***",overflowoffset=5
```

This format line defines the screen size to be 80 columns by 40 rows and specifies that the screen should be cleared before each page, that the overflow text should be flagged with the string *******, and that the overflow indentation should be 5 columns.

Related Information

The **ap**, **dp**, **mhl**, and **scan** commands in *AIX Operating System Commands Reference*.

“Overview of the Message Handling Package” in *Managing the AIX Operating System*.

Purpose

The message format for the Message Handling (MH) Package.

Description

The Message Handling (MH) Package processes messages in a particular format. While this format is different from that used by the Bell and Berkeley mail systems, the MH package can read Bell and Berkeley message files.

Each user has a mail drop that initially receives all messages processed by the **post** or **sport** command. The **inc** command reads from that mail drop and incorporates the new messages found there into the user's own mail folder (typically **inbox**). The mail drop consists of one or more messages.

Messages are expected to consist of lines of text. Graphics and binary data are not handled. No data compression is accepted. All text is ASCII 7-bit data.

The general memo framework of the ARPA Internet RFC 822 standard is used. A message consists of a block of information in a rigid format, followed by general text with no specified format. The rigidly formatted first part of a message is called the header, and the free-format portion is called the body. The header must always exist, but the body is optional. These parts are separated by an empty line, that is, two consecutive new-line characters. Within messages, the header and body can be separated by a line consisting of dashes:

To:
cc:
Subject

The header is composed of one or more header components. Each header component can be viewed as a single logical line of ASCII characters. If the text of a header component extends across several lines, the continuation lines are indicated by leading spaces or tab characters.

Each header component is composed of a keyword or name, along with associated text. The keyword begins at the left margin, cannot contain space or tab characters, cannot exceed 63 characters, and ends with a : (colon). Certain components (as identified by their keywords) must follow rigidly the defined formats in their text portions.

The text for most formatted components (for example, **Date:** and **Message-Id:**) is produced automatically. The text for other components must be entered by the user (for example,

To: and **cc:**). Multiple addresses are separated by commas. A missing host/domain is assumed to be the local host domain.

Header Components

Date:	Added by post , spost , or the mail transport system; contains the date and time of the message's entry into the transport system.
From:	Added by post or spost ; contains the address of the author or authors (can be more than one if a Sender field is present). Replies are typically directed to addresses in the Reply-To: or From: field (the former has precedence if both are present).
Sender:	Added by post or spost in the event that the message already has a From: line. This line contains the address of the actual sender. Replies are never sent to addresses in the Sender: field.
To:	Contains addresses of primary recipients.
cc:	Contains addresses of secondary recipients
Bcc:	Contains still more recipients. However, the Bcc: line is not copied into the message as delivered, so these recipients are not listed. The MH package uses an encapsulation method for blind copies.
Fcc:	Causes the post or spost command to copy the message into the specified folder for the sender, if the message was successfully given to the transport system.
Message-Id:	A unique message identifier added by post or spost if the -msgid flag is set.
Subject:	Sender's commentary. It is displayed by the scan command.
In-Reply-To:	A commentary line added by the repl command when replying to a message.
Resent-Date:	Added when redistributing a message by post or spost .
Resent-From:	Added when redistributing a message by post or spost .
Resent-To:	New recipients for a message resent by the dist command.
Resent-cc:	More recipients. See cc: and Resent-To: .
Resent-Bcc:	More recipients. See Bcc: and Resent-To: .
Resent-Fcc:	Copies resent message into a folder. See Fcc: and Resent-To: .
Resent-Message-Id:	A unique identifier attached by post or spost if you specify the -msgid flag. See Message-Id: and Resent-To: .

mh-mail

Resent: Annotation that **dist** uses when you specify the **-annotate** flag.
Forwarded: Annotation that the **forw** command uses when you specify the **-annotate** flag.
Replied: Annotation that **repl** uses when you specify the **-annotate** flag.

File

/usr/mail/\$USER Location of mail drop.

Related Information

Standard for the Format of ARPA Internet Text Messages, RFC 822.

“Overview of the Message Handling Package” in Managing the AIX Operating System.

mhook

Purpose

Specifies actions to be taken when mail is received.

Description

An *mhook* (or receive-mail hook) is an action that is automatically performed when new mail is received through the Message Handling (MH) Package. Whenever you receive a new message, the **sendmail** command searches for the file **.forward** in your **\$HOME** directory. **sendmail** pipes the new message to the **slocal** program when **\$HOME/.forward** exists and contains the following line:

```
| /usr/lib/mh/slocal
```

The **slocal** program reads the file **\$HOME/.maildelivery** and performs the actions specified in that file for each message being delivered. You can specify your own mail delivery instructions (or mhooks) in **\$HOME/.maildelivery**. Each line in **\$HOME/.maildelivery** describes an action and the conditions under which the action should be performed. Each line must contain five arguments separated by commas or space characters. These arguments are:

field *pattern* *action* *result* *string*

The following list describes each argument:

field Specifies a header component field to be searched for a match to the character string specified in the *pattern* argument. You can specify one of the following values for the *field* argument:

component Searches the specified header component.

*** Always matches.

addr Searches whatever field was used to deliver the message to you.

default Matches only if the message has not been delivered yet.

source Specifies the out-of-band sender information.

pattern Specifies the character string to search for in the header component given by the *field* argument. The *pattern* argument is not case sensitive. Thus, the character string matches any combination of uppercase and lowercase characters. You must specify a dummy pattern if you use *** or **default** in the *field* argument.

<i>action</i>	Specifies an action to take with the message if the message contains the pattern specified in the <i>pattern</i> argument. You can specify the following actions:
file or >	Appends the message to the file given by the <i>string</i> argument. If the message can be written to the file, the action is considered successful. When a message is appended to a file, the header component Delivery-Date: is added to the message to indicate when the message was appended to the file.
pipe or 	Pipes the message as standard input to the command named by the <i>string</i> argument, using the shell to interpret the string. If the exit status from the command is 0 (zero), the action is considered successful. Prior to giving the string to the shell, the string is expanded with the following built-in variables: <ul style="list-style-type: none"> \$(sender) The return address for the message. \$(address) The address that was used to deliver the message. \$(size) The size of the message in bytes. \$(reply-to) Either the Reply-To: or From: header component of the message. \$(info) Miscellaneous out-of-band information.
qpipe or ^	Similar to pipe , but executes the command directly after built-in variable expansion without assistance from the shell. If the exit status from the command is 0 (zero), the action is successful.
destroy	Always succeeds.
<i>result</i>	Indicates how the action should be performed. You can specify one of the following values for this argument: <ul style="list-style-type: none"> A Performed the action. If the action succeeds, the message is considered delivered. R Performs the action. Even if the action succeeds, the message is not considered delivered. ? Performs the action only if the message has not been delivered. If the action succeeds, the message is considered delivered.
<i>string</i>	If you use file as the <i>action</i> argument, <i>string</i> specifies the file to which the message can be appended. If you use pipe or qpipe , <i>string</i> specifies the command to execute. If you use destroy as the <i>action</i> argument, <i>string</i> is not used, but you must still include a dummy <i>string</i> argument.

All five arguments must be present in each line of the file. Blank lines in **.maildelivery** are ignored. Put a **#** (pound) character in the first column to indicate a comment.

If **.maildelivery** cannot be found, or does not deliver the message, **/usr/lib/mh/maildelivery** is used in the same manner. If the message is still not delivered, it is delivered to the user's maildrop, **/usr/mail/\$USER**.

MH contains four standard programs that can be run as receive-mail hooks: **rcvdist**, **rcvpack**, **rcvstore**, and **rcvttt**. *AIX Operating System Commands Reference* contains descriptions of these programs.

Example

The following example shows some lines that can be specified as mhooks in **\$HOME/.maildelivery**:

```
# If the message is from George, save it.
From      george          file      A          george.mail

# If the message is to the project manager, save a copy in log.
addr      manager        >          R          proj-X/statlog
# and forward it to Amy
addr      manager        |          A          "/usr/lib/mh/rcvdist amy"

# Save any messages not delivered.
default   -              >          ?          mailbox
```

Files

\$HOME/.forward	The file searched by the sendmail command when mail is received.
/usr/lib/mh/slocal	The slocal program that, when specified in
/usr/lib/mh/maildelivery	The mail delivery instructions. \$HOME/.forward , performs actions defined in \$HOME/.maildelivery .
\$HOME/.maildelivery	The file specifying receive-mail hooks for slocal to perform.

Related Information

“Overview of the Message Handling Package” in *Managing the AIX Operating System*.

The commands **rcvdist**, **rcvpack**, **rcvstore**, **rcvtty**, **sendmail**, and **slocal** in *AIX Operating System Commands Reference*.

mh-profile

Purpose

Customizes the Message Handling (MH) Package.

Description

Each user of the Message Handling (MH) Package is expected to have a file named **.mh-profile** in the home directory. This file contains a set of user parameters used by some or all of the MH programs. Each line of the file is in the following format:

profile-entry: value

Profile Entries

Of the possible profile entries, only **Path:** is required. The others are optional. Some entries have default values if the entries are not present. In the notation used in the following list, (profile, default) indicates whether the information is kept in the user's MH profile or context file, and the default value.

- | | |
|---------------------------|---|
| Path: | Specifies the location of the <i>user-mh-directory</i> directory. The usual location is \$HOME/Mail . (profile, no default) |
| context: | Declares the location of the MH context file. (profile, default: <i>user-mh-directory/context</i>) |
| Current-Folder: | Keeps track of the current open folder. (context, default: inbox) |
| Previous-Sequence: | Names the sequences that should be defined as the <i>msgs</i> or <i>msg</i> argument given to the program. If not present, or empty, no sequences are defined. Otherwise, for each name given, the sequence is first zeroed and then each message is added to the sequence. (profile, no default) |
| Sequence-Negation: | Defines the string which, when prefixed to a sequence name, negates that sequence. Thus, if Sequence-Negation: is set to not , then notseen means all those messages that are not a member of the sequence seen . (profile, no default) |
| Unseen-Sequence: | Names the sequences that are defined as those messages recently incorporated by the inc command. The show command removes messages from this sequence after they have been seen. If not present, or empty, no sequences are defined. Otherwise, for each |

mh-profile

	name given, the sequence is first zeroed and then each message is added to the sequence. (profile, no default)
mh-sequences:	Names the file in each folder that defines public sequences. To disable the use of public sequences, leave the value of this entry blank. (profile, default: .mh-sequences)
atr-seq-folder:	Keeps track of the private sequence named <i>seq</i> in the specified <i>folder</i> . (context, no default)
Editor:	Defines the editor to be used by the comp , dist , forw , and repl commands. (profile, default: prompter)
Msg-Protect:	Defines octal protection bits for message files. See the chmod command in <i>AIX Operating System Commands Reference</i> for an explanation of the octal number. (profile, default: 0644)
Folder-Protect:	Defines protection bits for folder directories. (profile, default: 0711)
program:	Sets default flags to be used whenever the specified MH <i>program</i> is invoked. For example, you can override the Editor: profile component when replying to messages by adding the profile entry: repl: -editor /bin/ed (profile, no defaults)
lasteditor-next:	Specifies the editor that is the default editor after using <i>lasteditor</i> . This takes effect at the What now? level of the comp , dist , forw , and repl commands. After editing the draft with <i>lasteditor</i> , the default editor is set to be <i>nexteditor</i> . If you enter edit without any arguments to What now? , then <i>nexteditor</i> is used. (profile, no default)
Folder-Stack:	Defines the contents of the folder stack of the folder command. (context, no default)
Alternate-Mailboxes:	Tells the repl and scan commands which addresses are really yours. In this way, repl knows which addresses should be included in the reply, and scan knows if the message really originated from you. Addresses must be separated by a comma, and the host names listed should be the official host names for the mailboxes you indicate, as local nicknames for hosts are not replaced with their official site names. For each address, if a host is not given, then that address on any host is considered to be you. In addition, an asterisk can appear at either or both ends of the mailbox and host to indicate wildcard matching. (profile, default: \$LOGNAME)

Draft-Folder:	Indicates a default draft folder for the comp , dist , forw , and repl commands. (profile, no default)
digest-issue-list:	Tells forw the last issue of the last volume sent for the digest <i>list</i> . (context, no default)
digest-volume-list:	Tells forw the last volume sent for the digest <i>list</i> . (context, no default)
MailDrop:	Tells inc your mail drop, if different from the default. This is superseded by the \$MAILDROP environment variable. (profile, default: /usr/mail/\$USER)
Signature:	Tells inc your mail signature. This is superseded by the \$SIGNATURE environment variable. (profile, no default)

The following profile elements are used whenever a MH program invokes another program. You can use **.mh-profile** to select alternate programs. The following list gives the default values.

fileproc:	/usr/bin/refile
incproc:	/usr/bin/inc
installproc:	/usr/lib/mh/install-mh
lproc:	/bin/pg
mailproc:	/usr/bin/mhmail
mhlproc:	/usr/lib/mh/mhl
moreproc:	/bin/pg
mshproc:	/usr/bin/msh
packproc:	/usr/bin/packf
postproc:	/usr/lib/mh/spost ¹
rmmproc:	none
rmfproc:	/usr/bin/rmf
sendproc:	/usr/bin/send
showproc:	/bin/pg
whatnowproc:	/usr/bin/whatnow
whomproc:	/usr/bin/whom

When you invoke a MH program, it reads the **.mh-profile** file by default. If you define the environment variable **\$MH**, you can specify another profile file. If the file of **\$MH** is not absolute (does not begin with a / (slash)), it will be presumed to start in the current directory. This is one of the few exceptions in the MH package where nonabsolute path names are not considered relative to your MH directory.

¹ The **spost** command uses the address handling capabilities of the **sendmail** command. If you do not have **sendmail** installed on your system, set the **postproc:** profile entry to **/usr/lib/post**.

mh-profile

Similarly, if you define the environment variable **\$MHCONTEXT**, you can specify a context other than the normal context file (as specified in the MH profile). If the value of **\$MHCONTEXT** is not absolute, it will be presumed to start from your MH directory.

MH programs also support the following other environment variables:

- \$MAILDROP** Tells **inc** the default mail drop. This supersedes the **MailDrop:** profile entry.
- \$SIGNATURE** Tells **send** and **post** your mail signature. This supersedes the **Signature:** profile entry.
- \$HOME** Tells all MH programs your home directory.
- \$TERM** Tells the MH package your terminal type. The **TERMCAP** variable is also consulted. In particular, these tell **scan** and **mhl** how to clear your terminal and how many columns wide your terminal is. They also tell **mhl** how many lines long your terminal screen is.
- \$editalt** Specifies an alternate message. This is set by **dist** and **repl** during edit sessions so that you can read the message being distributed or replied to. This message is also available through a link called **@** in the current directory if your current directory and the folder the message lives in are on the same AIX file system.
- \$mhdraft** Specifies the path of the working draft.
- \$mhfolder** Specifies the folder containing the alternate message. This is set by **dist** and **repl** during edit sessions so you can read other messages in the current folder besides the one being distributed. The **\$mhfolder** environment variable is also set by **show**, **prev**, and **next** for use by **mhl**.

Files

- | | |
|----------------------------------|--------------------------------------|
| \$HOME/.mh-profile | The user profile. |
| user-mh-directory/context | The user context file. |
| folder/.mh-sequences | Public sequences for <i>folder</i> . |

Related Information

In this book: “environment” on page 4-48.

The **mh** command in *AIX Operating System Commands Reference*.

“Overview of the Message Handling Package” in *Managing the AIX Operating System*.

mh-tailor

Purpose

Defines how MH commands work.

Description

The entries located in the file `/usr/lib/mh/mtstailor` tailor how MH commands work.

File Entries

The following list describes the `/usr/lib/mh/mtstailor` file entries and their default values. All of the file entries are optional.

File Entry	Description
------------	-------------

localname:	Specifies the host name of the local system. If this entry is not defined, MH queries the system for the default value.
-------------------	---

systemname:	Specifies the host name of the local system in the uucp domain. If this entry is not defined, MH queries the system for the default value.
--------------------	--

mmdfldir:	Specifies the location of mail drops. If this entry is present and empty, mail drops are located in the user's \$HOME directory. If this entry does not exist, mail drops are located in the <code>/usr/mail</code> directory.
------------------	---

mmdflfil:	Specifies the name of the file used as the maildrop. If this entry is not defined, the default file name is the same as user name.
------------------	--

mmdelim1:	Specifies the beginning-of-message delimiter for mail drops. The default value is -001-001-001-001-n.
------------------	---

mmdelim2:	Specifies the end-of-message delimiter for mail drops. The default value is -001-001-001-001-n.
------------------	---

mmailid:	Specifies whether support for MMail IDs in <code>/etc/passwd</code> is enabled. If mmailid: is set to a nonzero value, support is enabled. The pw-gecos field in the password file has the form:
-----------------	--

My Full Name *mailid*

When support for MMail IDs is enabled, the internal MH routines that deal with user and full names return *mailid* and **My Full Name** respectively. The default value is 0 (zero).

mh-tailor

- lockstyle:** Specifies the locking discipline. The value 1 (one) creates lock names by appending **.lock** to the name of the file being locked. The default value is 1.
- lockldir:** Specifies the directory for locked files. The default value is **/etc/locks**.
- sendmail:** Specifies the path name of the **sendmail** program. The default value is **/usr/lib/sendmail**.
- maildelivery:** Specifies the path name of the file containing the system default mail delivery instructions. The default value is **/usr/lib/mh/maildelivery**.
- everyone:** Specifies the users to receive messages addressed to everyone. All users having UIDs greater than the specified number (not inclusive) receive messages addressed to everyone. The default value is 200.

Files

/usr/lib/mh/mtstailor The MH tailor file.

Related Information

The **ap**, **conflict**, **inc**, **msgchk**, **msh**, **post**, **rcvdist**, and **rcvpack** commands in *AIX Operating System Commands Reference*.

“Overview of the Message Handling Package” in *Managing the AIX Operating System*.

mnttab

Purpose

Provides a table of mounted file systems.

Synopsis

```
#include <mnttab.h>
```

Description

The **mnttab** file contains a table of devices, mounted by the **mount** command. Each **mnttab** file entry has the following structure:

```
struct mnttab {  
    char  mt_dev[100];  
    char  mt_filsys[100];  
    short mt_ro_flg;  
    time_t mt_time;  
};
```

The fields indicate the following:

mt_dev The null-padded name of the mounted special file.

mt_filsys The null-padded name of the mounted-on directory.

mt_ro_flg This flag indicates if the file system is mounted read-only. A value other than 0 indicates the file system is mounted read-only.

mt_time The time the file system was mounted.

The **mountab** parameter, which is defined while the system is being customized, determines the number of simultaneously mounted file systems and therefore the maximum number of entries in the **/etc/mnttab** file. This parameter changes when the **mountab** parameter in the **master** file changes.

mnttab

File

`/etc/mnttab`

Related Information

In this book: “filesystems” on page 3-74, “master” on page 3-125, and “system” on page 3-210.

The **config**, **mount**, and **umount** commands in *AIX Operating System Commands Reference*.

netgroup

Purpose

Lists network groups.

Description

The **netgroup** file lists specific groups defined within a network. It is used in the permission checking process when remote mounts, remote logins and remote shells are executed. The information in **netgroup** is used to classify machines for remote mounts, and users for remote logins and remote shells. Network groups maps for the Yellow Pages data bases are created from the **/etc/netgroup** file. (See the Files Section later in this discussion.)

Each line of the **netgroup** file defines a group in the following format:

groupname member1 member2 ... to membern

The *member* entries can be another group name, or a triple (*hostname, username, domainname*). In the triple, any of the fields can be empty to signify a wild card. For example, a group to which all users belong can be defined in **netgroup** as follows:

universal (,,)

If the name in one of the fields starts with something other than a letter, digit or underscore, the entry works in the exact opposite way. For example, a - symbol used in the following:

justmachines (mach1,-,readers)

means that the machine mach1 belongs to the group justmachines, but no users belong to the group.

Files

/etc/netgroup
/etc/yp/netgrp usr/netgroup.dir
/etc/yp/domainname/netgrp.pag
/etc/yp/domainname/netgrp.usr.dir
/etc/yp/domainname/netgrp.user.pag
/etc/yp/domainname/netgrp.hst.dir
/etc/yp/domainname/netgrp.hst.pag

Related Information

The **makedbm** and **ypserv** commands in *AIX Operating System Commands Reference*.

The Yellow Pages section in *Managing the AIX Operating System*.

options

Purpose

Defines the valid choices for each **ddi** option.

Description

The **/etc/ddi/options** file contains a sorted list of the valid choices for each keyword used in **ddi** files. The **devices** command uses this file to display the valid choices for the keywords during the **add**, **change**, and **showdev** subcommands.

Each line must follow the following format:

optionchoices

where:

<i>option</i>	This field is exactly 20 characters long, is padded on the right with spaces, and contains no tab characters. An <i>option</i> is one of the following:
<i>keyword</i>	The keyword for which the valid choices are to be specified
<i>keywordadapter</i>	The keyword followed by the adapter name
<i>keywordclass</i>	The keyword followed by the device class
<i>keywordtype</i>	The keyword followed by the device type
<i>keywordstanza</i>	The keyword followed by the name of the device stanza in the system file.

The **devices** command looks for one of these combinations based on the setting of the **opts** keyword in the **kaf** file for the device. See “kaf” on page 3-121 for details about the **opts** keyword.

choices This field is exactly 29 characters long, is padded on the right with spaces, and contains no tab characters.

Note: The **/etc/ddi/options** file must be sorted alphabetically by the *option* field. If it is not sorted, then the **devices** command displays incorrect information about the options available for a given keyword.

The use of extended characters in the **/etc/ddi/options** file is not supported.

options

File

`/etc/ddi/options`

Related Information

In this book: “ddi” on page 3-47, “descriptions” on page 3-65, and “kaf” on page 3-121.

passwd

Purpose

Contains passwords.

Synopsis

```
#include <pwd.h>
```

Description

The **passwd** file is an ASCII file that contains all the information that defines a user on the system. The **passwd** file contains:

- Login name.
- ! (exclamation point). A placeholder indicating that the encrypted password is in the **/etc/security/passwd** file.
- Numerical user ID.
- Numerical group ID.
- Additional data for each user.
- Initial current directory.
- Program to use as shell.

Each field is separated from the next by a colon. The file has general read permission, but the security-relevant information is stored in the **/etc/security/password** file. Therefore, a user can use the file to map numerical user IDs to names without potentially compromising the security of other users.

The **users** command maintains this file. Programs use the **getpwent** subroutines to extract various fields in this file.

If the user password field is **NULL** in **/etc/security/password**, the user has no password and the user password field is empty. Encrypted passwords are stored in the **/etc/security/passwd** file.

If the program field is **NULL**, the shell (**/bin/sh**) is used. The program field can contain parameters passed when the **exec** system call is issued. Parameters are separated by a space (such as a space or tab characters). A **** (backslash) is used for escapement when a

parameter contains a space. The **login** command accepts the program name and as many as 14 parameters. Any more than 14 parameters are ignored. A maximum of 4096 characters can be used for the program name and its parameters. More than 4096 characters causes **login** to exit. Parameters in this field can use symbolic escapement for the following special characters: **\n**, **\r**, **\v** (produces 013), **\b**, **\t**, and **\f**. Also, **\0** through **\7** builds a 1-byte octal number. Anything else that is preceded with a **** (backslash) passes through.

The format of the additional data for each user is:

```
full_name | file_limit ; site_info
```

where:

full_name	Contains the name of the user whose 8-character (or fewer) login name is the first field.
file_limit	Specifies the maximum length of files created by the user. See the login command in <i>AIX Operating System Commands Reference</i> and the ulimit system call.
site_info	Available for user information that may be required by applications at a specific installation. Can contain any printable character other than a colon.

Any or all of the subfields can be omitted. If the *file_limit* subfield is omitted, the preceding **/** (slash) is omitted and the system default limit is used. If the *site_info* subfield is omitted, the preceding **;** (semi-colon) is also omitted.

Passwords

The encrypted password is 13 characters long. The characters used come from the extended characters (code page P0, see “data stream” on page 4-5) and can be uppercase or lowercase characters, numerals, and the **.** (dot) and **/** (slash) characters except when the password is null. In this case, the encrypted password is also null. For more information about password characteristics see the discussion of password security in *Managing the AIX Operating System*.

Password security is provided by two files: **/etc/passwd** paired with **/etc/security/passwd**. The encrypted password field of **/etc/passwd** contains a **!** (exclamation point) as a placeholder. The encrypted password is contained in the **passwd** field of **/etc/security/passwd**.

The following information is contained in a stanza for the user in the **/etc/security/passwd** attribute file. Only a superuser has access to this file.

- Encrypted password
- Audit class
- Lastupdate
- Restrictions

The **users** command maintains these files. Programs use the **getpwent** subroutines to extract various fields from these files, and the **putpw** subroutine to update the fields for a user.

The format for an entry in **/etc/passwd** is:

```
login : ! : uid : gid : info : home : program
```

These fields are described as follows:

login	The login name of the user. This must also be the name of a stanza in /etc/security/passwd . If such a stanza does not exist, sysck creates the stanza.
!	A placeholder indicating that the encrypted password is stored in the /etc/security/passwd file.
uid	The user ID. It must be a decimal numeral.
gid	The primary group ID. It must be a decimal numeral.
info	The additional data for each user.
home	The user's home directory.
program	The program that runs when the user logs in to the system.

The format of the user stanza in **/etc/security/passwd** is:

```
login:
    password = <encrypted string>
    lastupdate = <date>
    restrictions = <comma-separated string>
    auditclasses = <comma-separated string>
```

The field definitions for the **login** stanza are as follows:

password	The user's encrypted password. If this field is missing or NULL , the user has no password.
lastupdate	The date on which the user last changed the password. Password aging is controlled on a system-wide basis by attributes in the password stanza of /etc/security/config .
restrictions	The constraints imposed on passwords. See the discussion of password security in <i>Managing the AIX Operating System</i> .
auditclasses	The initial audit class for the user (see "auditevents" on page 2-36). The audit command updates this field.

Japanese Language Support Information

Password security is not available when Japanese Language Support is installed on your system.

passwd

Files

`/etc/security/config`
`/etc/passwd`
`/etc/security/passwd`

Related Information

In this book: “a64l, l64a” on page 2-13, “crypt, encrypt” on page 2-116, “group” on page 3-113, “getpwent, getpwuid, getpwnam, setpwent, endpwent” on page 2-376, “putpw, putpwent, putpwsent” on page 2-579, and “ulimit” on page 2-835.

The **login**, **passwd**, and **users** commands in *AIX Operating System Commands Reference*.
“Overview of International Character Support” in *Managing the AIX Operating System*

plot

Purpose

Provides the graphics interface.

Description

The subroutines described in “plot” produce output files of the format outlined in this section. The **tplot** commands interpret these graphics files for various devices, performing the plotting instructions in the order in which they appear.

A graphics file consists of a stream of plotting instructions. Each instruction consists of an ASCII letter usually followed by bytes of binary information. A point is designated by 4 bytes representing the *x* and *y* values; each value is a 2-byte signed integer. The last designated point in an **l**, **m**, **n**, or **p** instruction becomes the current point for the next instruction.

The following table lists each of the **plot** instructions and the corresponding **plot** subroutines.

Instr	Sub	Description
a	arc	Draws the arc described by the following 12 bytes. The first 4 bytes describe the center point (<i>x</i> , <i>y</i>) of the arc or circle. The second 4 bytes describe the beginning point of the arc. The third 4 bytes describe the ending point of the arc. Arcs are drawn counterclockwise. The results are unpredictable if the three points do not really form an arc.
c	circle	Draws a circle whose center point is defined by the first 4 bytes, and whose radius is given as an integer in the following 2 bytes.
e	erase	Starts another frame of output.
f	linemod	Uses the following string, terminated by a new-line character, as the style for drawing further lines. The styles are dotted, solid, long-dashed, short-dashed, and dot-dashed.
l	line	Draws a line from the point designated by the next 4 bytes to the point designated by the following four bytes.

plot

Instr	Sub	Description
m	move	The next 4 bytes designate a new current point.
n	cont	Draws a line from the current point to the point designated by the next 4 bytes.
p	point	Plots the point designated by the next 4 bytes.
s	space	The next 4 bytes designate the lower left corner of the plotting area; followed by 4 bytes for the upper right corner. The plot is magnified or reduced to fit the device as closely as possible.
t	label	Places the following ASCII string so that its first character falls on the current point. A new-line character terminates the string.

The space setting

```
space(0, 0, 480, 432);
```

exactly fills the plotting area with unity scaling for the IBM Personal Computer Graphics Printer. The upper limit is immediately outside the plotting area, which is taken to be square. Points outside the plotting area can be displayed on devices that do not have square displays.

Related Information

In this book: “plot” on page 2-549 and “TERM” on page 4-76.

The **graph** and **tplot** commands in *AIX Operating System Commands Reference*.

ports

Purpose

Describes the ports.

Description

The **ports** file contains the names and characteristics of all the system terminal ports. It provides a convenient means to associate values with named keyword parameters on a port-by-port basis. Defaults are supplied as a user requests. Some **ports** attributes are active after you run the **secure** command.

The **getty** process is the principal user of the information in this file. Since programs using this file look for specific keyword parameters and ignore all others, parameters other than those discussed here can be added to this file as necessary.

File Format

The **ports** file consists of one or more named stanzas usually separated by blank lines. Each stanza begins with its name followed by a colon, and contains assignments of values to keyword attributes. The values, in turn, can be alphanumeric strings or arbitrary character strings enclosed in double quotation marks.

Stanzas headed by the name **default** specify attribute-value pairs that are associated with all of the ports following it to the next **default** stanza. Explicit values within a port stanza override this association.

Port-Control Parameters

Most of the parameters in the **ports** file are port controls for login terminals. Because there are system defaults specified in the **getty** process, it is usually not necessary to specify more than a few attributes in the **ports** file, as shown in the example. The port control parameters are described as follows:

enabled The **init** program uses this attribute to determine whether or not to create a logger on the port. If the port should permit a logger, the value can be **TRUE**, **SHARE**, or **DELAY**. Otherwise the value is **FALSE**. Normally the value **TRUE** is used to enable the port. However, if the port is shared (bidirectional use) then the value should be **SHARE** or **DELAY**. The value **SHARE** makes the port bidirectional with the device-locking scheme used by **uucp**, **cu**, **ate**, and **connect**. **DELAY** operates like **SHARE** except that the **getty** process must read one or more characters from the port before the login herald will print. **DELAY** is useful with direct connections and intelligent

	modems. Note that penable , pdisable , and phold commands override the specified value.
eof	An octal integer specifying the character code that causes an end-of-file character to generate from the terminal. The system default is 004 (or 0x04), the ASCII EOF character, which Ctrl-D generates.
eol	An optional and seldom used alternate line termination character to use in addition to the ASCII new-line (line feed) character.
erase	An octal integer specifying the character code that deletes the previously received character. The system default for the erase character is 010 (or 0x08), Ctrl-H , which is generated on many terminals by the Backspace key.
herald	An arbitrary string, enclosed in double quotation marks, printed by the getty process to prompt for login. The C language \ (backslash) escapes \r , \n , \t , \b , and \f are recognized as carriage return, new-line, tab, backspace, and form feed, respectively.
imap	This attribute is used by getty to set the terminal input map. If imap is not specified, getty resets the map to the system default.
intr	An octal integer specifying the character code that interrupts the running process. The system default is 0177 (or 0x7f), which is usually generated by a key labeled Del or Rubout .
kill	An octal integer specifying the character code that deletes the input line. The system default for the kill character is 025 (or 0x15), Ctrl-U , which is the ASCII NAK character.
lock	This attribute requests port locking. If the value is TRUE , init creates a file in /etc/locks when the port is enabled and deletes the lock file when the port is disabled. Similarly, penable does not enable a port whose lock attribute is TRUE when the corresponding lock file exists. Programs using the port for some other purpose (such as a link between processors) should check for an outstanding lock (and create a lock file, if necessary) before opening the port.
logger	A character string giving the names the program is to use at login. The default is /bin/login .
logmodes	Console modes in effect while prompting for and reading in the user name. Modes are specified as a series of terminal options separated by a + (plus). Terminal options are as listed in the stty command. All listed modes not preceded with - (dash) are recognized. For example, the default logmodes parameter is specified as:

```
logmodes = cread+cs8+hupcl+echoe+echok
```

Because a speed value is not recognized in **logmodes** under any circumstances, the baud rate must be set with the **speed** parameter (see below).

min	See the discussion of ICANON under “termio” on page 5-133.
omap	This attribute is used by getty to set the terminal output map. If omap is not specified, getty resets the map to the system default.
owner	Normally, when a port is logged in, the login program sets the logged-in user to be the owner of that port. By specifying an owner, either a UID or user name, the system manager forces the getty process to set ownership even before opening the port.
parity	The values ODD , EVEN , and NONE cause the generation of odd, even, and no parity, respectively, while INPACK , IGNPAR , and PARMRK cause the checking input for parity errors, ignoring input characters with parity errors, and “marking” input parity errors as specified under “termio” on page 5-133. These values can be combined, as in <code>parity=odd+inpck</code> .
program	If specified, this parameter is the name of the program to run immediately after setting the logmodes. This parameter establishes purpose server ports that respond to a connection with a special protocol handler. If the special assignment program=HOLD is specified, no program runs on the port, but the logmodes, ownership, and protection are set and the port is held open. This keeps the desired modes associated with a port that is occasionally used for some special purpose.
protection	Normally the protection on a terminal is set to rw--w--w- (octal 622 or 0x192). The protection parameter overrides this default. The value can be set to an octal mask or a string, such as rw-rw-rw- (octal 666 or 0x1b6).
quit	An octal integer specifying the character code that causes the running process to abort. The system default is 026 (or 0x16), which is generated by pressing Ctrl-V .
runmodes	Console modes in effect after the user name is read. The mode in which the port is left, specified similar to logmodes .
sak	<p>This attribute is used by getty to determine whether sak processing is enabled on the port. In a secure system, sak processing should be enabled on every interactive port. It may be necessary to disable the sak on ports used for inter-machine communication.</p> <p>After you run the secure command, <code>sak = on</code>.</p> <p>The sak attribute is not available when Japanese Language Support is installed on your system.</p>
shell	<p>This attribute is a program that runs as the user shell after login, regardless of the user’s preference. This attribute is used in trusted path management. The user shell is passed as an argument to the shell program, which forces actman to run on the hft.</p> <p>After you run the secure command, <code>shell = /bin/actman</code>.</p>

- The **shell** attribute is not available when Japanese Language Support is installed on your system.
- speed** A decimal integer from the set {50, 75, 110, 134.5, 150, 300, 600, 1200, 1800, 2000, 2400, 3600, 4800, 7200, 9600, 19200} depending on the hardware capability.
- synonym** The **init** and **login** commands use **synonym** to assure that all special files which refer to the same device receive the same protection. The value of this attribute is an alternate name for the **dev/console** stanza.
- After you run the **secure** command, **synonym** = /dev/hft.
- The **synonym** attribute is not available when Japanese Language Support is installed on your system.
- term** This parameter is passed to the logger and shell in their environment (see “environment” on page 4-48) in the variable **TERM**. Some application software uses this information to determine the type of terminal the user is using.
- time** See the discussion of **ICANON** under “termio” on page 5-133.
- timeout** A decimal integer. If a user name is not specified before the given number of seconds, the **getty** process advances to the next port setting, or exits if all settings were exhausted.

Multiple values, separated by commas, can be specified as in the **speed = 300,1200** line for dial-in terminals. This causes the port to be set up according to the first set of values for each attribute. If a framing error occurs, as a result of a user-generated **BREAK** on the line or a speed mismatch between the terminal and the set speed, the **getty** process advances to the next value on the list.

If multiple specifications occur for more than one parameter, all advance at the same time. Thus, a specification such as the following first tries the line at 300 baud with no parity:

```
speed=300,1200
parity=none,odd+inpck
```

If a framing error occurs, it tries 1200 baud generation and checks for odd parity.

Other Port Parameters

The other parameters in the file are:

- loc** The location of the terminal connected to the port. This parameter is presently unused by any RT software. Because the programs that access this file ignore keywords they do not use, you can add helpful information here to keep all port-specific information together.
- printer** The hard-copy device used for output from optional word processing packages.

Example

The following example of a **ports** file illustrates some of its features:

```
default:
    enabled = false
    speed = 9600
    herald = "\033[H\033[J\rRT(noname)\r\nlogin: "
    printer = lp0
    term = dumb
    erase = 010
    kill = 025
    intr = 0177

/dev/console:
    loc = "console"
    term = hft
    enabled = true
    herald = "\033[H\033[J\rRT(/dev/console)\r\nlogin:"
```

Files

etc/ports
etc/locks

Related Information

In this book: “attributes” on page 3-20, “connect.con” on page 3-37, “environment” on page 4-48, and “termio” on page 5-133.

The **su**, **pstart**, **pdisable**, **getty**, **login**, **init**, and **stty** commands in *AIX Operating System Commands Reference*.

“Overview of International Character Support” in *Managing the AIX Operating System*.

portstatus

portstatus

Purpose

Centralizes commands to control ports.

Description

The **pstart**, **penable**, **pshare**, **pdelay**, **pdisable**, and **phold** commands communicate with the **init** command (the process that controls loggers) through the **/etc/portstatus** file. See the **pstart** and **pdisable** commands in *AIX Operating System Commands Reference* for more information on these commands. The format of this file is:

```
struct portstatus
{  char ps_line[14];          /* device name */
    char ps_stat;             /* current status */
    char ps_rqst;             /* requested status */
};

#define ENABLE    01          /* spawn logger */
#define DISABLE   02          /* kill logger */
#define HOLD      04          /* spawn no new logger */
#define SHARE     010
#define DELAY     020
```

The fields are explained as follows:

ps_line Names the special file for the port, **tty01**, for example.

ps_rqst Used by the **pstart**, **penable**, **pshare**, and **pdelay** commands to request changes in the enabling of a port.

ps_stat Used by the **init** command to show the current state of the port.

The **pstart**, **penable**, **pshare**, and **pdelay** commands, with the **-i** flag specified, automatically initializes this file. If this file does not exist or is damaged, the **init** command cannot enable the ports properly.

File

/etc/portstatus

Related Information

The **pstart** and **init** commands in *AIX Operating System Commands Reference*.

predefined

predefined

Purpose

Provides information for predefined devices.

Description

The **predefined** file contains information about hardware adapters and devices that is used by the **devices** command. Some of these devices may not be present in a particular configuration, but all of them are supported by the system. The **predefined** file contains information needed when adding one of these devices so that you do not have to supply the information yourself. The size of this file increases with new entries as additional licensed programs are installed in the system.

The **devices** command uses the information in this file to set up stanzas in the **system** and **qconfig** files when devices are added to the system. Note that information in this file has no effect on the system until it is moved to a stanza in the **system** or **qconfig** file.

The **predefined** file is similar in structure and content to the **system** file, and its stanzas can contain any of the keywords that are allowed in the **system** file. See “system” on page 3-210 for a description of the keywords that can appear in stanzas of these files.

The use of extended characters in the **predefined** file is not supported.

The **predefined** file contains several special stanzas:

defqueue	Used by the devices command to create the queue stanza in the qconfig file when a printer or plotter is added.
defdevice	Used by the devices command to create the device stanza in the qconfig file when a printer or plotter is added.
default	Contains keywords and their values that are common to all device stanzas.
adpts	Contains a list of adapters with the code number and description that the devices command uses to identify each one.
addr	Contains a list of adapters and their corresponding adapter type and address.

Example

The following shows sample entries of the **predefined** file.

defqueue:

argname = none
device = none

defdevice:

file = /dev/none
backend = /usr/lpd/piobe

default:

modes = rw-rw-rw-
owner = root

adpts:

mp = 49
* IBM Mono Disp & Paral Prntr
sp1 = 23
* IBM Ser/Par Adptr, Primary
sp2 = 23
* IBM Ser/Par Adptr, Secondary
rs232c1 = 35

addrs:

2A03BC = mp
2A0378 = sp1par
2A0278 = sp2par
2303F8 = sp1
2302F8 = sp2
351230 = rs232c1

5182:

* IBM PC Color Printer (5182)
name = 5182
nname = 5182
driver = u5182
crname = true

predefined

```
    minor = c
    vint = 4
    iodn =
    kaf_file = /etc/ddi/pprinter.kaf
    kaf_use = kparallel
    file = /etc/ddi/pprinter
    use = d5182
    noddi = false
    dtype = printer
* Printer
    switchable = true
* Coprocessor Device
    specproc = cfgaqcfg
    shared = false
    noduplicate = false
    dname = lp
    noshow = false
    aflag = true
* adapter description
    adp = mp,sp1,sp2
```

File

/etc/predefined

Related Information

In this book: “attributes” on page 3-20 and “system” on page 3-210.

profile

Purpose

Sets the user environment at login time.

Description

The **profile** file contains commands to be executed at login and variable assignments to be set and exported into the environment. The **/etc/profile** file contains commands executed by all users at login.

After the **login** program adds the **LOGNAME** (login name) and **HOME** (login directory) parameters to the environment, the commands in **\$HOME/.profile** are executed, if it is present. The **.profile** file is the individual user profile that overrides the variables set in the **profile** file and is used to tailor the user environment variables set in **/etc/profile**. The **.profile** file is often used to set exported environment variables and terminal modes. The person who customizes the system can use **adduser** to set default **.profile** files in each user home directory. Users can tailor their environment as desired by modifying their **.profile** file.

Example

The following example is typical of a **/etc/profile** file:

```
# Set file creation mask
unmask 022
# Tell me when new mail arrives
MAIL=/usr/mail/$LOGNAME
# Add my /bin directory to the shell search sequence
PATH=/bin:/usr/bin:/etc::
# Set terminal type
TERM=hft
# Make some environment variables global
export MAIL PATH TERM
```

profile

Files

`$HOME/.profile`
`/etc/profile`
`/etc/profile.dos`

Related Information

In this book: “environment” on page 4-48 and “TERM” on page 4-76.

The **env**, **login**, **mail**, **sh**, **stty**, and **su** commands in *AIX Operating System Commands Reference*.

qconfig

Purpose

Configures a printer queueing system.

Description

The **/etc/qconfig** file describes the queues and devices available for use by the **print** command, which places requests on a queue, and the **qdaemon** command, which removes requests from the queue and processes them. The **/etc/qconfig** file is an attribute file. See “attributes” on page 3-20 for details about the format of attribute files.

Some stanzas in this file describe queues, and other stanzas describe devices. Every queue stanza requires that one or more device stanzas immediately follow it in the file. The first queue stanza describes the default queue. The **print** command uses this queue when it receives no queue parameter.

The name of a queue stanza must be 1 to 3 characters long. The following table shows some of the field names along with some of the possible values that appear in this file:

acctfile	Identifies the file used to save print accounting information. FALSE , the default, indicates suppress accounting. If the named file does not exist, no accounting is done.
argname	Identifies the queue name identifier that is used in the print command to specify the queue.
CD	Determines the handling of spool files. TRUE indicates qdaemon will change the directory to the file directory, and translate user and group IDs. FALSE , the default, turns this feature off.
device	Identifies the symbolic name that refers to the device stanza.
discipline	Defines the queue serving algorithm. The default, fcfs , means first come first served. sjn means shortest job next.
friend	Indicates whether the backend updates the status file and responds to terminate signals. TRUE is the default. FALSE indicates it does not.
up	Defines the state of the queue. TRUE , the default, indicates that it is running. FALSE indicates that it is not running.

If a field is omitted, its default value is assumed. The default values for a queue stanza are:

```
friend      = TRUE
discipline  = fcfs
up          = TRUE
acctfile    = FALSE
CD          = FALSE
```

Also, the default **argname** value is the name of the stanza preceded by a - (dash). The **device** field cannot be omitted.

The name of a device stanza is arbitrary. The fields that can appear in it stanza are:

access	Specifies the type of access the backend has to the file specified by the file field. The value of access is write if the backend has write access to the file, or both if it has both read and write access. This field is ignored if the file field has the value FALSE .
align	Specifies whether the backend sends a form-feed control before starting the job if the printer was idle. The default is FALSE .
backend	Specifies the full path name of the backend, optionally followed by flags and parameters to be passed to it.
feed	Specifies the number of separator pages to print when the device becomes idle, or the value NEVER , which indicates that the backend is not to print separator pages.
file	Identifies the special file where the output of backend is to be redirected. FALSE , the default, indicates no redirection. In this case, the backend opens the output file.
header	Specifies whether a header page prints before each job or group of jobs. NEVER , the default, indicates no header page at all. ALWAYS means a header page before each job. GROUP means a header before each group of jobs for the same user.
trailer	Specifies whether a trailer page prints after each job or group of jobs. NEVER , the default, means no trailer page at all. ALWAYS means a trailer page after each job. GROUP means a trailer page after each group of jobs for the same user.

The **qdaemon** places the information contained in the **feed**, **header**, **trailer**, and **align** fields into a status file that is sent to the backend. Backends that do not update the status file do not use the information it contains.

If a field is omitted, its default value is assumed. The **backend** field cannot be omitted. The default values in a device stanza are:

```
file      = FALSE
access    = write
feed      = never
header    = never
trailer   = never
align     = FALSE
```

The **print** command automatically converts the ASCII **qconfig** file to binary when the binary version is missing or older than the ASCII version. The binary version is found in **/etc/qconfig.bin**.

Unlike the format of the **ports** file, the **qconfig** file format does not allow default stanzas.

Examples

1. The batch queue supplied with the AIX system might contain these stanzas:

```
bsh:
    argname = bsh
    friend  = FALSE
    discipline = fcfs
    device  = bshdev
```

```
bshdev:
    backend = /bin/sh
```

To run a shell procedure called **myproc** using this batch queue, type:

```
print bsh myproc
```

The queuing system runs the files one at a time, in the order submitted. The **qdaemon** process redirects standard input, standard output, and standard error to the **/dev/null** file.

qconfig

2. To allow two batch jobs to run at once:

```
bsh:
    argname      = save
    friend       = FALSE
    discipline   = fcfs
    device       = bsh1,bsh2
```

```
bsh1:
    backend      = /bin/sh
```

```
bsh2:
    backend      = /bin/sh
```

Files

```
/etc/qconfig
/etc/qconfig.bin
/usr/lpd/digest
```

Related Information

In this book: “attributes” on page 3-20.

The **print**, **lp**, and **qdaemon** commands in *AIX Operating System Commands Reference*.

rasconf

Purpose

Defines the reliability, availability, and serviceability (RAS) configuration file.

Description

The **rasconf** file defines attributes of the reliability, availability, and serviceability (RAS) system. Initially, RAS logging is inactive and must be activated before any RAS data can be collected.

This attribute file consists of stanzas that govern the actions of daemons associated with individual RAS devices. Each stanza name is the name of the associated RAS device.

The following attributes are valid:

- buffer** = *size* Specifies the size of a buffer in 1024-byte blocks. You use this buffer to catch RAS trace events. The default size is six.
 Note: If the buffer is too small, error messages appear in your trace file. When this occurs increase the buffer size.
- file** = *file* Specifies the file into which the daemon will write the RAS information.
- size** = *blocks* Specifies the maximum size, in 1024-byte blocks, to which the daemon will allow the file to grow.

Example

A typical **rasconf** file can contain the following:

```
/dev/osm:
  file = /usr/adm/ras/osm
  size = 100

/dev/error:
  file = /usr/adm/ras/errfile
  size = 50
```

rasconf

```
/dev/trace:
file = /usr/adm/ras/trcfile
size = 80
buffer = 6
```

File

/etc/rasconf

Related Information

In this book: “attributes” on page 3-20, “error” on page 5-31, “osm” on page 5-124, and “trace” on page 5-150.

The **errdemon** and **trace** commands in *AIX Operating System Commands Reference*.

RPC

Purpose

Describes the RPC program numbers.

Description

The **RPC** file describes the RPC program numbers in text so that users can identify the number with the name. The names identified in the text can be used in place of RPC program numbers.

Each line of the file contains the following:

- Name of the server for the RPC program
- RPC program number
- Aliases of the RPC program.

The names are in character text so users can read them. The items are separated by blank spaces or tab characters. A # (pound) character designates the start of a comment. Routines that search this file do not interpret the text contained from the # to the end of the line.

Example

The following shows an example of an **rpc** file:

portmapper	100000	portmap sunrpc
rstatd	100001	rstat rup perfmeter
rusersd	100002	rusers
nfs	100003	nfsprog
ypserv	100004	ypprog
mountd	100005	mount showmount
ypbind	100007	
walld	100008	rwall shutdown
yppasswdd	100009	yppasswd
etherstadd	100010	etherstat
rquotad	100011	rquotaprog quota rquota
sprayd	100012	spray

RPC

3270_mapper	100013	
rje_mapper	100014	
selection_svc	100015	selnsvc
database_svc	100016	
rex	100017	rex
alis	100018	
sched	100019	
llockmgr	100020	
nlockmgr	100021	
x25.inr	100022	
statmon	100023	
status	100024	

File

`/etc/rpc`

Related Information

In this book: “Remote Procedure Call (RPC)” on page 2-615.

The section on NFS in *Managing the AIX Operating System*.

The section on RPC in *AIX Operating System Programming Tools and Interfaces*.

sendmail.cf

Purpose

Contains **sendmail** configuration file data.

Description

The configuration file contains the configuration information for the **sendmail** program. The configuration information includes such items as the host name and domain and the **sendmail** rule sets.

The configuration file has three major purposes:

- To initialize the environment for **sendmail** by setting the options
- To rewrite addresses in messages by first mapping the addresses from any format into a canonical form and then mapping the canonical form into the appropriate syntax for the receiving mailer.
- To translate the address into the set of instructions needed to deliver the message.

The configuration file entries consist of lines, each of which begins with a single character command. Entries can continue onto multiple lines by placing white space at the beginning of each subsequent line. Comments are included on lines beginning with the # (pound) character. The commands and operands are:

CX *word1 word2 ...*

Defines the class, specified by *X*, of words to match on the left hand side of rewriting rules. Class specifiers may be any of the uppercase letters from the ASCII character set. Lowercase letters and special characters are reserved for system use.

DX *value*

Defines the macro specified by *X* and its associated *value*. A macro is named using a single character. The character can be any character from the ASCII character set, but user-defined macros can only use the uppercase letters. Lowercase letters and special characters are reserved for system use. Macros can be interpolated in most places using the escape sequence *\$x*. See "Special Macros" on page 3-200 for additional information.

FX *filename [format]*

Reads the elements of the class specified by *X* from *filename* using an optional **scanf** format specifier.

H[?mflags?]hdrname: htemplate

Defines the header format the **sendmail** program inserts into a message. Continuation lines are a part of the definition and write into the outgoing message. The *htemplate* is macro-expanded before insertion into the message. If the *mflags* are specified, at least one of the specified flags must be stated in the mailer definition for this header to be automatically output. If one of these headers is in the input, the header writes into the output message regardless of these flags.

Mname, [field=value]*

Defines a mailer where *name* is the name of the mailer (used internally only) and *field=value* defines attributes of the mailer. Allowable *fields* and *values* are:

P=value Defines the path name of the mailer, where *value* is the path name. SMTP internal mail uses a value of **IPC**.

F=value Defines the special flags for this mailer, where *value* can be a string composed of the following:

- C** Saves the sender domain. For this function only, the sender domain ID defined to be everything from the first @ (at) character to the end of the sender address. This string is appended to header addresses which contain no @ whenever mail is sent to any mailer. This also applies to calculation of the **\$g** macro and everything dependent on it. This flag offsets the SMTP **mail** and **rcpt** commands.
- D** Requires a **Date** header line.
- e** This mailer is expensive to connect to. Avoid normal connection and when necessary connect during a queue run. See the **c** option on 3-196.
- E** Escape **From** lines to >**From** in message bodies.
- f** Calls the mailer with an **-f** flag followed by the expression of **\$g**, if the specified mailer is legitimate and the user ID of the **sendmail** process is on the trusted-user list or is root or, if the group ID is system. Else, the **-f** flag on the mailer call is not generated.
- F** Requires a **From** header line.

- h** Preserves uppercase in host names for this mailer.
- I** Uses SMTP to communicate with another **sendmail** and can use special protocol features.
- l** This mailer is local; final delivery is performed.
- L** Limits the line length of a text line to less than 1000 characters. Any leading dot duplicated due to the **X** flag is not included in the count. Only allows 7-bit data to pass either way through the mailer.
- m** This mailer can send to multiple users on the same host in one transaction. When a **\$u** macro occurs in the **A** part of the mailer definition, that field will be repeated as necessary for all qualifying users.
- M** Requires a **Message-ID** header line.
- n** The AIX-style From line on the front of the message is not inserted.
- N** International Character Support. Only has meaning when used with the **L** flag. Allows 8-bit data to pass.
- p** Uses the *return-path* in the SMTP **MAIL FROM:** command rather than just the return address.
- P** Requires a **Return-Path** header line.
- r** Same as option **f** except a **-r** flag is generated.
- s** Strips quotation marks off of the address before calling the mailer.
- S** User ID is not reset before calling the mailer.
- u** Preserves uppercase in users names for this mailer.
- U** Requires **From** lines with UUCP-style **remote from** *host* on the end.
- x** Requires a **Full-Name** header line.
- X** This mailer uses the hidden-dot algorithm. (Any line beginning with a dot has an extra dot prepended.) This ensures that the lines in the message containing a leading dot will not terminate the message

prematurely. See the **sendmail -i** flag or the **config** option.

- S = value** The rewriting rule set to be used for sender addresses, where *value* is the rewriting rule set number.
- R = value** The rewriting rule set to be used for recipient addresses, where *value* is the rewriting rule set number.
- A = arg** Defines the argument string *arg* to exec the mailer with. Embedded spaces may be included. If embedded spaces are used, enclose the argument string with " (double quotation marks). For a SMTP mailer, **A = IPC**.
- E = string** Defines the *string* to use as an end-of-line indication. A string containing only **newline** is the default.
- M = length** Defines the maximum message *length* to be sent to the mailer.

Ox[value]

Sets option to *x*. If the option is a valued option, you must also specify *value*. Options may also be selected from the command using the **-o** flag of the **sendmail** command. The options and the possible values are described as follows:

- Afile** Uses the named *file* as the alias file.
- Bc** Sets the blank substitution character to the character specified in the parameter *c*. The **sendmail** program replaces unquoted spaces in addresses with this character. The supplied configuration file uses the . (dot) for this character.
- c** If an outgoing mailer is marked as expensive to use, this option causes **sendmail** to queue messages for that mailer program without sending them. The queue can be run later when costs are lowest or when the queue is large enough to send the message efficiently.
- dx** Sets the delivery mode to *x*. Valid modes are:
 - b** Deliver in background (asynchronously). This is the default setting.
 - i** Deliver interactively (synchronously)

- q**

Queue the message only and deliver during queue run.
- ex**

Sets error processing to mode *x*. Valid modes are:

 - e** Mails the error message to the user's mail box, but always exits with a zero exit status (normal return).
 - m** Mails the error message to the user's mail box.
 - p** Displays the error message on the terminal (default).
 - q** Discards the error message and returns the exit status only.
 - w** Writes the error message to the terminal or mails it if the user is not logged in.
- f**

Saves **From** lines at the front of messages. These lines are normally discarded. Causes all other headers to be regarded as part of the message body.
- gN**

Sets the default group ID to use when calling mailers to the value specified by *N*.
- Hfile**

Specifies the name of the SMTP help file.
- i**

Does not interpret a . (dot) on a line by itself as a message terminator. Removes the excess dot inserted by a remote mailer at the beginning of a line, if mail is received through SMTP. In addition, if receiving mail through SMTP, any dot at the front of a line followed by another dot is removed. This is the opposite of the action performed by the **X** mailer flag.
- Ix**

Allows spaces as well as tabs to separate the LHS and RHS of rewrite rules. In both the LHS and RHS, *x* must be used in place of embedded spaces. The default for *x* is _ (underscore). All instances of *x* are changed to spaces after the LHS and RHS are separated by the **sendmail** program. This option allows rewrite rules to be modified using an editor that replaces tabs with spaces.
- Ln**

Specifies the log level to be the value supplied in the *n* parameter. Each number in the following list includes the activities of all numbers of lesser value and adds the activity that it represents.

Valid levels and the activities they represent are:

- 0 No logging
- 1 Major problems only
- 2 Message collections and failed deliveries
- 3 Successful deliveries
- 4 Messages being deferred
- 5 Placing messages in the queue
- 6 Unusual but benign incidents
- 9 Log internal queue ID to external message ID mappings
- 12 Several messages that are of interest when debugging
- 16 Verbose information regarding the queue.

m If the sender is in an alias expression, also send to the sender.

Mx value Defines macro *x* to have *value*. This option is normally used only from the **sendmail** command line.

n Validates the RHS of aliases when performing the **newaliases** function.

Nnetname Sets the name of the host network to *netname*. The **sendmail** program compares the argument of an SMTP **HELO** command to *hostname.netname* (value of *hostname* comes from the kernel). If these values do not match, it adds the *hostname.netname* string to the **Received:** line in the message so that messages can be traced accurately.

o Indicates that this message can have old style headers. Without this option, the message has new style headers (commas instead of spaces between addresses). If this option is set, an adaptive algorithm correctly determines the header format in most cases.

Paddress Identifies the person who is to receive a copy of all returned mail.

Qdir Sets the directory in which to queue messages. The directory will be created if it does not exist.

<i>rtime</i>	Sets the time out for reads from a mailer program to the value specified by <i>time</i> . If no time out value is set, sendmail waits indefinitely for a mailer to respond.
<i>Sfile</i>	Sets the mail statistics file to the <i>file</i> . Statistics are only collected if the file exists. This file must be created by the user.
s	Enqueues before delivery, even when in immediate delivery mode.
<i>Ttime</i>	Sets the time out on messages in the queue to the specified <i>time</i> . After this interval, sendmail returns the message to the sender. The default is three days.
uN	Sets the default user ID to use when calling mailers to the value specified by <i>N</i> .
v	Run in verbose mode.
Y	The sendmail program delivers each message in the mail queue from a separate process. This option is not required and can increase overhead in the AIX environment.
Pname = num	Defines values for the Precedence field. When <i>name</i> is found in a Precedence field, the message class is set to <i>num</i> . Higher numbers indicate higher precedence. Negative numbers indicate that error messages are not returned. The default <i>num</i> is 0. The precedence of mail is defined by a header of that name within the mail.
Rlhs rhs comments	Defines a rewriting rule. One or more tab characters separate the three fields of this command. If space characters are to be used, the configuration option I must be set. The fields may contain embedded spaces, unless the I option is set. If the I option is set, the embedded spaces must be represented by the character defined in I . After the fields are separated, the character representing the space is changed to an actual space.
Sx	Sets the rule set currently collected to number <i>x</i> . If a rule set definition is begun more than once, the new definition overwrites the old.
Tuser1 user2...	Defines system-administrative (trusted) user IDs. These IDs have permission to override the sender address using the -f flag. There can be more than one ID specified per line.

Special Macros

Macros are interpolated using the construct `$x`, where `x` is the name of the macro to be interpolated. Lowercase letters are reserved to have special semantics, used to pass information in or out of the **sendmail** program.

The following macros must be defined to transmit information into the **sendmail** program:

- e** The SMTP entry message. This message is sent by the SMTP handler in the **sendmail** domain when the host connects to it.
- j** The official domain name for the site. This must be the first word in the **\$e** macro. The domain name is a sequence of domain element strings, ordered from the most specific to the most general, separated by periods. The use of nicknames or aliases is not allowed. The maximum domain name length is 64 characters. The **\$j** macro should use this format.
- l** The format of the AIX **From** line. This macro is usually a constant.
- n** The name of the daemon (for error messages). This macro is usually a constant.
- o** The set of operators in addresses. This macro consists of a list of characters considered to be tokens and separates tokens during parsing. For example, if **r** exists in the **\$o** macro, the input, address, parses into three tokens: **add**, **r**, and **ess**. There are many internal hard-coded delimiters added to this list by **sendmail**. It is recommended that this list not be changed.
- q** The default format of the sender address.

Sendmail defines some macros for interpolation into argument variables for mailers or for other contexts. These macros are:

- a** The origination date in Arpanet form. **\$a** contains the time extracted from the **Date** line of the message (if there is one). If the incoming message has no **Date:** line, the **\$a** macro contains the current time.
- b** The current date in Arpanet form. **\$b** equals the current date and time (used for postmarks).
- c** The **hop count**. The hop count is the number of times the message has been processed. The **-h** flag of the command line or the number of **Received:** headers in the message determine the hop count.
- d** The date in AIX (**ctime**) format.
- f** The sender (from) address. The **\$f** macro is the sender address as seen from the current host.
- g** The sender address relative to the receiver. When mailing to a specific host, the **\$g** macro contains the address of the sender relative to the receiver. For example, if the user, newton, at system, appletree, sends a message to chopin@piano, the **\$f** macro equals newton and the **\$g** macro equals newton@appletree.
- h** The receiving host.
- i** The queue ID of the host. The **\$i** macro is useful for tracking messages if put into the message ID line.
- p** The process ID of **sendmail**. **\$p** and **\$t** are used to create unique strings for the **Message-ID** field.
- s** The host name of the sender.
- t** A numeric representation of the current time. The macros, **\$p** and **\$t**, are used to create unique strings for the **Message-ID** field.
- u** The receiving user.
- v** The version number of AIX. The **\$v** macro can be found in **Received:** header messages and is useful for debugging.
- w** The host name of the local site and, if present, the address.
- x** The full name of the sender. The name is determined by one of the following: the full name passed as a flag to **sendmail**, the value found in the **Full-Name** line of the header, the value found in the comment field of a **From** line, or if the message originates locally, the full name found in **/etc/passwd**.
- y** The terminal ID of the sender.
- z** The home directory of the receiver.

sendmail.cf

Files

- | | |
|-----------------------------------|---|
| <code>/usr/lib/sendmail.cf</code> | The sendmail configuration file. |
| <code>/usr/lib/sendmail.fc</code> | The compiled version of the sendmail configuration file. |

Related Information

Managing the AIX Operating System.

The command **sendmail** in *AIX Operating System Commands Reference*.

serverattach

Purpose

Describes the code service state for an RT system.

Description

The code service attribute file describes the code service state for a particular system. It is a standard attribute file in directory */etc/codeserve*. The **serverattach** attribute file primarily controls:

- Active service client processes
- State conditions and transitions for any system.

The file contains a stanza for every possible server for this node. If this node has no servers, the stand-alone stanza takes effect. The file usually processes serially, from the first stanza to the last. In the following order, the stanzas describe the primary server, alternate servers, the stand alone parameter, and the default stanza.

The command **chnstate** controls the attribute file processing. The **chnstate** parameters permit the following process control over the attribute file:

- If there are no parameters, each stanza processes serially, beginning with the primary stanza, until a server attaches or the system becomes a stand-alone system.
- The command can specify a starting stanza, as a parameter. The system ignores previous stanzas, but processes normally the stanzas that follow the starting stanza.
- Exclude a stanza from the process. The process must be repeated for each stanza to be excluded.
- The system may override incorrect values of the following passed parameters. The new value of an overridden parameter applies to all stanzas.
 - **mode**
 - **time**
 - **interval**

Each stanza describes the target state for this particular node as either active or stand-alone. If the stanza specifies **active**, it also indicates the relationship between a client node and its server.

The *servername* can be either a nickname or a node ID. The first stanza identifies the primary server. Subsequent stanzas identify alternate servers followed by the default

serverattach

stanza. The last stanza is the stand-alone stanza and should not include a *servername* entry.

The valid keywords and values are:

- mode** Indicates how the client should attempt to achieve compatibility with a server operating in the **active** code service state. Possible values are:
- auto** System will interact with the client's user as little as possible.
 - manual** System will not upgrade a licensed program or another program without interacting with the user.

If a user on a client operating in the active code service state wants to prevent automatic upgrades of licensed programs or programs, the value should be set to **MANUAL**. The default stanza forces a mode of **MANUAL** if mode is not specified.
- state** The code service state in which a system runs when attached to a particular server. Possible values are **active** and **standalone**.
- time** The value of **time** is the maximum number of seconds that a client will attempt to attach to a server in either **active** or **historyvalidation** modes.

Note: The **historyvalidation** mode is a temporary process where the client ensures that it is fully compatible with the programs installed on the server. If the client cannot attach to the server within the specified time, the client attempts to execute the instructions in the next stanza, if present. If the user specifies a time of less than 60 seconds, then the system forces a value of 60 seconds.
- interval** Equals the delay time, in seconds, between efforts to attach to the server in either **active** mode or **historyvalidation** mode. If **interval** is not specified, the default stanza sets an interval time of 10 seconds.

Example

* Primary server stanza

```
servera:  
  mode = auto  
  state = active  
  time = 60  
  interval = 15
```

* Backup server stanza

```
serverb:  
  mode = manual  
  state = active  
  time = 90  
  interval = 10
```

* Default system stanza

```
default:  
  state = standalone
```

```
standalone:  
  state = standalone  
  mode = manual  
  time = 60  
  interval = 10
```

File

/etc/codeserve/serverattach

Related Information

In this book: “attributes” on page 3-20.

The **chngstate** command in *AIX Operating System Commands Reference*.
Managing the AIX Operating System.

sysck.cfg

Purpose

Describes the system's security-relevant files.

Description

The `/etc/security/sysck.cfg` file contains the names and characteristics of security-relevant files in the system. It describes trusted programs, their configuration files, and their data files. The `sysck` command interprets the contents of `/etc/security/sysck.cfg`.

The `sysck.cfg` file is an attribute file (see "attributes" on page 3-20). Each stanza describes a security-relevant file. The attributes of this file appear within the stanza for the file.

All attributes are optional and are described as follows:

- class** Files grouped by function. A system administrator can direct `sysck` to check the description of all files with a specific **class**.
- owner** File owner ID. This value can be the login name of a user in `/etc/passwd` or a decimal numeral.
- group** File group ID. This value can be the name of a group in `/etc/group` or a decimal numeral.
- mode** File mode (type and permission bits). This value can be an octal number or a string of the form:

```

- - - - -
b r w x r w x r w x
c      s s t
d      S      S      T
p

```

where the string consists of one letter from each column.

links Full path names that are linked to this file.

symlinks Full path names that are symbolically linked to this file.

sysck.cfg

- tcb** File tcb attribute. This value can be **TRUE** or **FALSE**.
- program** Full path name of a program that performs additional consistency checks on the file. Arguments to the program also appear with this attribute. Note that the value of an attribute must be enclosed in double quotes if it has embedded spaces. The **sysck** command will execute this program with any arguments preceded by the same flags (**-n**, **-p**, or **-y**) as were provided on the **sysck** command line.

Example

The following example of a **sysck.cfg** stanza shows some of its features:

```
/etc/passwd:
    class =          passwd
    owner =           root
    group =           system
    tcb =             true
    mode =            -rw-r--r--
    program =         "/etc/security/pwdchk"
```

```
/etc/security/passwd:
    class =          passwd
    owner =           root
    group =           system
    tcb =             true
    mode =            -rw-----
```

```
/dev/lp0
    class =          device
    owner =           root
    group =           printq
    mode =            crw-rw----
    links =           /dev/lp,/dev/LP
```

File

/etc/security/sysck.cfg

Related Information

In this book: “attributes” on page 3-20.

The **sysck** commands in *AIX Operating System Commands Reference*.

system

system

Purpose

Identifies the system devices.

Description

The **system** file contains entries for currently configured real devices, virtual devices, and device managers. Some of this information is used by the VRM to establish the default running environment.

The **system** file is an attribute file containing stanzas that generally describe special files and includes information about device drivers. See "attributes" on page 3-20 for a description of attribute files. Also included is data for the **Define-Device** supervisor calls (SVCs) to the VRM if needed. See the **vrconfig** program in *AIX Operating System Commands Reference* for more information.

Each special file named in the **system** file refers to a device driver entry in the **master** file. The driver entries specify the kernel device drivers to be configured. All drivers needed for specified special files are included, and those drivers marked as mandatory. Two driver entries may specify the same major device number where the same driver controls devices with different I/O code numbers.

The name of each stanza is the simple name of the special file.

The use of extended characters in the **system** file is not supported.

- | | |
|--------------|---|
| adp | Indicates the valid adapter names for the device described in the stanza. |
| aflag | <p>Is a required keyword that indicates whether both an adapter and a kernel device driver are associated with the device described in the stanza. If the value is FALSE, the device has either a kernel device driver or a VRM device driver (and, in most cases, an adapter), but not both.</p> <p>If TRUE, the devices command constructs the name of the ddi stanza when adding the device by concatenating the value of the use keyword, the adapter name that the user chooses from the adp list, and a port number.</p> <p>If FALSE, devices constructs the name of the ddi stanza by concatenating the value of the use keyword and a pseudo port number. Either a maxminor keyword or a maxdev keyword must be defined if the value of aflag is false. If the maxminor keyword is defined, which indicates that this device has only a kernel device driver then the pseudo port number is the next unused integer between 0 and maxminor - 1. If</p> |

the **maxdev** keyword is defined, which indicates that this device has only a VRM device driver, then the pseudo port number is the next unused integer between 0 and **maxdev** - 1.

The **showall** subcommand of the **devices** command displays the comment line that immediately follows the **aflag** definition as a description of the device or adapter.

crname	Is a required keyword that indicates by a value of TRUE or FALSE whether the devices command should create a new driver name for the device.
ddi	Specifies the hexadecimal bytes to be passed to the VRM Define_Device SVC. If a customize helper program is invoked, it determines the data to be passed. In that case, this attribute is not used.
dname	Indicates the prefix name that is used to create the name of the device stanza in the /etc/system file and the special file in the /dev directory. The devices command uses this value when it creates a stanza name for a new special file.
driver	Identifies the associated driver in the master file. This is mandatory in all device stanzas.
dtype	Specifies the class of the device. Examples of this are printer and disk . The devices command displays this value when asking the user to choose a device class. It also uses this value to construct a list of device classes.
file	Identifies the file that contains the stanzas included by the use attribute. This is the /etc/ddi file associated with the device.
iodn	Specifies the I/O device number to use. If omitted, no Define_Device SVC is sent to the VRM.
kaf_file	Indicates the name of the keyword attribute file to be used by the customization helper programs for the device described in the device stanza.
kaf_use	Indicates the name of the stanza in the kaf_file that contains information about the attributes for the device.
maxdev	Specifies the maximum number of IODNs supported by the VRM device driver for this adapter. This is equivalent to the maximum number of adapters that can be installed times the number of ports on each adapter.
minor	Has a value of the form cn , where c is either b to denote a block device, or c to denote a character device. n is the minor device number.
modes	Sets the protection bits for the special file, specified in the form rwrxrwxrwx . Hyphens replace modes that are turned off, for example, rw-r--r-- .

system

name	Is a required keyword that identifies the type of the driver or the 4-character name passed to the VRM using the Define_Device SVC. The default name is the first two and last 2 characters of the special file name.
native	Identifies the model. A value TRUE indicates IBM RT 6150, FALSE indicates IBM RT 6151.
nname	Is a required keyword that indicates the name of the device, adapter, and port number (if applicable).
nocopy	The value TRUE indicates that the associated VRM device driver stanza in /etc/master cannot contain the copy keyword, but must specify the code keyword instead.
noddi	Indicates whether any device-dependent information is associated with the device. The value TRUE indicates there is none. If noddi=TRUE , then the change subcommand of the devices command does not allow the user to change device characteristics.
nodelete	Indicates whether to delete the special file when this driver is removed. When this value is TRUE , no attempt is made to delete the special file.
nodl	Indicates whether the device can be deleted from the system by the devices command. The value TRUE indicates the device cannot be deleted using this command.
noduplicate	Indicates whether another device of this type can be added to the system. The value TRUE indicates another device cannot be added.
noipl	Indicates whether this stanza is processed at initial program load (IPL) time. When this value is TRUE , this stanza is not processed at system initial program load (IPL) time.
noshow	Indicates whether the devices command displays information from the stanza to the user. If noshow=FALSE , then the showdev subcommand of the devices command displays all device characteristics and the showall subcommand displays the device.
nospecial	When this value is TRUE , no special file (/dev file) is to be created.
owner	Specifies the name of the owner assigned to the /dev special file when it is created.
port	Lists the number of ports on each adapter in the adp keyword. There is a one to one correspondence between each adapter and its number of ports. If the device being added is the adapter, port is the number of ports on the adapter. This keyword is required if the aflag keyword is true.
protocol	Indicates whether this stanza is used by a protocol procedure. When this value is TRUE , this stanza is used by a protocol procedure.

shared	Sets the shared bit for the VRM Define_Device SVC. When this value is TRUE , the shared bit is set.
specproc	Indicates the name of the special processing routine that is to be invoked when customizing the system for the device. See “cfgadev” on page 2-64 for information about the application program interface to this feature.
switchable	Indicates whether the device can be shared by the coprocessor. The value TRUE indicates that it can be shared. If so, then the devices command displays this as a device that can be added to the coprocessor.
type	Defines a device manager rather than a device driver when this value is manager and used with the Define_Device SVC.
uinfo	Specifies the hexadecimal bytes to pass to the CFUDRV type ioctl call to configure a kernel device driver. If a customization helper program is invoked, it determines the data to pass. In that case, this attribute is not used.
use	Identifies a stanza to be logically included in the current stanza. If a file attribute is present, the file is searched to find the indicated stanza for device-dependent information. This keyword is required if the file keyword is present.
vdmgr	Defines device drivers controlled by a manager. Values are the names of stanzas in the system file, separated by commas. The controlled drivers should include nospecial=true .
vint	Identifies the virtual interrupt level to use. Level 4 is the only supported level. If not specified, the default value is vint=4 .

Other parameters can be given for special **customization helper** programs.

Miscellaneous System Parameters

Both the **master** and the **system** files can have option lines in the **default** stanzas describing miscellaneous system customizing and tuning options. Options in the **system** file override those in the **master** file. See “master” on page 3-125 for a list of these parameters.

Other lines can be added as needed.

system

Example

The following is an excerpt of the **system** file entries:

```
* * system - actual devices
```

```
default:
```

```
    modes = rw-rw-rw-  
    owner = root  
    native = true
```

```
lp1:
```

```
* IBM PC Color Printer
```

```
    name = 5182  
    crname = true  
    minor = cl  
    vint = 4  
    iodn = 12003  
    kaf_file = /etc/ddi/pprinter.kaf  
    kaf_use = kparallel  
    file = /etc/ddi/pprinter  
    noddi = false  
    dtype = printer  
    nodelete = true
```

```
* Printer
```

```
    switchable = true  
    specproc =cfgaqcfg  
    shared = false  
    noduplicate = false  
    dname = lp  
    noshow = false  
    aflag = true
```

```
* IBM Mono Disp & Paral Prntr
```

```
    adp = mp,sp1,sp2  
    use = d5182mp  
    nname = 5182mp  
    driver = u5182mp
```

```
lp2:
* IBM PC Color Printer (5182)
    name = 5182
    crname = true
    minor = c2
    vint = 4
    iodn = 12004
    kaf_file = /etc/ddi/pprinter.kaf
    kaf_use = kparallel
    file = /etc/ddi/pprinter
    noddi = false
    dtype = printer
```

```
* Printer
    switchable = true
    specproc = cfgaqcfg
    shared = false
    noduplicate = false
    dname = lp
    noshow = false
    aflag = true
```

```
* IBM Ser/Par Adptr, Primary
    adp = mp,spq,sp2
    use = d5182sp1
    nname = 5182sp1
    driver = u5182sp1
```

```
lp3:
* IBM PC Color Printer (5182)
    name = 5182
    crname = true
    minor = c3
    vint = 4
    iodn = 12005
    kaf_file = /etc/ddi/pprinter.kaf
    kaf_use = /etc/ddi/pprinter
    noddi = false
    dtype = printer
```

```
* Printer
```


system

```
switchable = true
specproc = cfgaqcfg
shared = false
noduplicate = false
dname = lp
noshow = false
aflag = true
* IBM Ser/Par Adptr, Secondary
adp = mp,sp1,sp2
use = d5182sp2
nname = 5182sp2
driver = u5182sp2
```

File

`/etc/system`

Related Information

In this book: “attributes” on page 3-20, “ddi” on page 3-47, “master” on page 3-125, and “predefined” on page 3-180.

The **config**, **devices**, and **vrmlconfig** commands in *AIX Operating System Commands Reference*.

tar

Purpose

Describes the tape archive format.

Description

The **tar** command reads and writes tapes in tape archive format. A **tar** tape consists of several 512-byte logical blocks that can be grouped (on magnetic tape) into records, which are some constant multiple of 512-byte blocks long. Block in the following description means logical block.

The following is the format of a file header that precedes each disk file written on the tape:

```
struct {
    char name[100];
    char mode[8];
    char uid[8];
    char gid[8];
    char size[12];
    char mtime[12];
    char chksum[8];
    char linkflag;
    char linkname[100];
};
```

All fields, except **linkflag**, are ASCII null-terminated strings. Numeric fields can contain leading blanks. The fields have the following meanings:

chksum	Contains a byte-by-byte sum of the entire header block assuming that the chksum field is all blanks.
gid	Contains the group identification of the file, in octal.
linkflag	Contains a 1 (one) if this is file is a link to a previous file on the tape; otherwise null.
linkname	Contains the name of a file if linkflag has a value of 1. The file named in this field is linked to the name file.
mode	Contains the mode of the file, which includes the protection bits, setuid bits, setgid bits, and file type, in octal.

tar

mtime	Contains the modification time, in octal. This field gives the major/minor device number for special files.
name	Contains the name of the file.
size	Contains the size in bytes, in octal. This field is 0 for special files.
uid	Contains the user identification of the file, in octal.

Unused bytes are null. Following the file header block are the data blocks of the file. The last block is null-padded if necessary. Two null blocks designate the end of the tape.

Directories and special files are treated in a slightly different way. A directory size is 0, meaning no data blocks follow, and its name ends with a / (slash). A special file is also written with 0 size. Its major/minor device number is in the **mtime** field.

Related Information

The **tar** command in *AIX Operating System Commands Reference*.

terminfo

Purpose

Describes terminals by capability.

Description

A **terminfo** file is a data base that describes terminals, defining their capabilities and their methods of operation. It is used by various programs, including the Extended Curses Library (**libcur.a**) and the **vi** editor. The information defined includes initialization sequences, padding requirements, cursor positioning, and other command sequences that control specific terminals.

This section explains the **terminfo** source file format. Before a **terminfo** source file can be used, it must be compiled using the **tic** command, which is described in *AIX Operating System Commands Reference*. You can edit and modify these source files, such as **/usr/lib/terminfo/ibm.ti**, which describes IBM terminals, and **/usr/lib/terminfo/dec.ti**, which describes DEC terminals.

See "TERM" on page 4-76 for a list of some of the terminals supported by predefined **terminfo** data base files and the corresponding values for the **TERM** environment variable.

Each **terminfo** entry consists of a number of fields separated by commas, ignoring any white space between commas. The first field for each terminal gives the various names the terminal is known separated by | (vertical bar) characters. The first name given should be the most common abbreviation for the terminal, the last name given should be a long name fully identifying the terminal, and all others are understood as synonyms for the terminal name. All names except the last should be in lowercase and not contain blanks. The last name can contain uppercase characters for readability.

Terminal names (except the last) should be chosen using the following conventions. A root name should be chosen to represent the particular hardware class of the terminal. This name should not contain hyphens, except to avoid synonyms that conflict with other names. Possible modes for the hardware or user preferences are indicated by appending a - (hyphen) and an indicator of the mode to the root name. Thus, a terminal in 132 column mode would be *term-w*.

terminfo

The following suffixes should be used where possible:

Suffix	Meaning	Example
-am	With automatic margins (usually default)	<i>term-am</i>
-c	Color mode	<i>term-c</i>
-w	Wide mode (more than 80 columns)	<i>term-w</i>
-nam	Without automatic margins	<i>term-nam</i>
-n	Number of lines on the screen	<i>term-60</i>
-na	No arrow keys (leave them in local)	<i>term-na</i>
-np	Number of pages of memory	<i>term-4p</i>
-rv	Reverse video	<i>term-rv</i>

Types of Capabilities

Capabilities in **terminfo** are of three types: Boolean, numeric, and string. Boolean capabilities indicate that the terminal has some particular feature. Boolean capabilities are true if the corresponding name is in the terminal description. Numeric capabilities give the size of the terminal or the size of particular delays. String capabilities give a sequence that can be used to perform particular terminal operations.

Entries can continue onto multiple lines by placing white space at the beginning of each subsequent line. Comments are included on lines beginning with the # (pound) character.

List of Capabilities

The following table shows **VARIABLE**, which is the name the programmer uses to access the **terminfo** capability. The **CAP NAME** (capability name) is the short name used in the text of the data base, and is used by a person updating the data base. The **I. CODE** is the 2-letter internal code used in the compiled data base, and always corresponds to a **termcap** capability name.

Capability names have no absolute length limit. An informal limit of 5 characters is adopted to keep them short and to allow the tabs in the source file **caps** to be aligned. Whenever possible, names are chosen to be the same as or similar to the ANSI X3.64 standard of 1979.

- (P) Indicates that padding can be specified.
- (G) Indicates that the string is passed through **tparm** with parameters as given (**#i**).
- (*) Indicates that padding can be based on the number of lines affected.
- (**#i**) Indicates the *i*th parameter.

VARIABLE	CAP NAME	I. CODE	DESCRIPTION
Booleans:			
auto-left-margin	bw	bw	Indicates cub1 wraps from column 0 to last column.
auto-right-margin	am	am	Indicates terminal has automatic margins.
beehive-glitch	xsb	xs	Indicates a terminal with F1 = Esc and F2 = Ctrl-C .
ceol-standout-glitch	xhp	xs	Indicates standout not erased by overwriting.
eat-newline-glitch	xenl	xn	Ignores new-line character after 80 columns.
erase-overstrike	eo	eo	Erases overstrikes with a blank.
generic-type	gn	gn	Indicates generic line type (such as dialup, switch)
hard-copy	hc	hc	Indicates hard-copy terminal.
has-meta-key	km	km	Indicates terminal has a meta key (shift, sets parity bit).
has-status-line	hs	hs	Indicates terminal has extra status line.
insert-null-glitch	in	in	Indicates insert mode distinguishes nulls.
memory-above	da	da	Retains information above display in memory.
memory-below	db	db	Retains information below display in memory.
move-insert-mode	mir	mi	Indicates safe to move while in insert mode.
move-standout-mode	msgr	ms	Indicates safe to move in standout modes.
over-strike	os	os	Indicates terminal overstrikes.
status-line-esc-ok	eslok	es	Indicates escape can be used on the status line.
teleray-glitch	xt	xt	Indicates destructive tabs and blanks inserted while entering standout mode.
tilde-glitch	hz	hz	Indicates terminal cannot print ~ (tilde) characters.
transparent-underline	ul	ul	Overstrikes with underline character.
xon-xoff	xon	xo	Indicates terminal uses xon/xoff handshaking.
Numbers:			
columns	cols	co	Specifies the number of columns in a line.
init-tabs	it	it	Provides tabs initially every # spaces.
lines	lines	li	Specifies the number of lines on screen or page
lines-of-memory	lm	lm	Specifies the number of lines of memory if > lines. A value of 0 indicates a variable.

terminfo

VARIABLE	CAP NAME	I. CODE	DESCRIPTION
magic-cookie-glitch	xmc	sg	Indicates number of blank characters left by sms or rm .
padding-baud-rate	pb	pb	Indicates lowest baud where carriage return and line return padding is needed.
virtual-terminal	vt	vt	Indicates virtual terminal number.
width-status-lines	wsl	ws	Specifies the number of columns in status line.

Strings:

appl_defined_str	apstr	za	Application-defined terminal string.
back-tab	cbt	bt	Back tab. (P)
bell	bel	bl	Produces an audible signal (bell). (P)
box_chars_1	box1	bx	Box characters primary set.
box_chars_2	box2	by	Box characters alternate set.
box_attr_1	batt1	Bx	Attributes for box_chars_1 .
box_attr_2	batt2	By	Attributes for box_chars_2 .
carriage-return	cr	cr	Indicates carriage return. (P*)
change-scroll-region	csr	cs	Changes scroll region to lines 1 through 2. (PG)
clear-all-tabs	tbc	ct	Clears all tab stops. (P)
clear-screen	clear	cl	Clears screen and puts cursor in home position. (P*)
clr-eol	el	ce	Clears to end of line. (P)
clr-eos	ed	cd	Clears to end of the display. (P*)
color-bg_0	colb0	d0	Background color 0 black.
color-bg_1	colb1	d1	Background color 1 red.
color-bg_2	colb2	d2	Background color 2 green.
color-bg_3	colb3	d3	Background color 3 brown.
color-bg_4	colb4	d4	Background color 4 blue.
color-bg_5	colb5	d5	Background color 5 magenta.
color-bg_6	colb6	d6	Background color 6 cyan.
color-bg_7	colb7	d7	Background color 7 white.
color-fg_0	colf0	c0	Foreground color 0 white.
color-fg_1	colf1	c1	Foreground color 1 red.
color-fg_2	colf2	c2	Foreground color 2 green.
color-fg_3	colf3	c3	Foreground color 3 brown.
color-fg_4	colf4	c4	Foreground color 4 blue.
color-fg_5	colf5	c5	Foreground color 5 magenta.

VARIABLE	CAP NAME	I. CODE	DESCRIPTION
color-fg-6	colf6	c6	Foreground color 6 cyan.
color-fg-7	colf7	c7	Foreground color 7 black.
column-address	hpa	ch	Sets cursor column. (PG)
command-character	cmdch	CC	Indicates terminal command prototype character can be set.
cursor-address	cup	cm	Indicates screen relative cursor motion row #1 col #2. (PG)
cursor-down	cud1	do	Moves cursor down one line.
cursor-home	home	ho	Moves cursor to home position (if no cup).
cursor-invisible	civis	vi	Makes cursor invisible.
cursor-left	cubl	le	Moves cursor left one space.
cursor-mem-address	mrcup	CM	Indicates memory relative cursor addressing.
cursor-normal	cnorm	ve	Makes cursor appear normal (undo vs or vi).
cursor-right	cuf1	nd	Indicates nondestructive space (Cursor Right).
cursor-to-ll	ll	ll	Moves cursor to first column of last line (if no cup).
cursor-up	cuu1	up	Moves cursor up one line (Cursor Up).
cursor-visible	cvvis	vs	Makes cursor very visible.
delete-character	dch1	dc	Deletes character. (P*)
delete-line	dll	dl	Deletes line. (P*)
dis-status-line	dsl	ds	Disables status line.
down-half-line	hd	hd	Indicates subscript (forward 1/2 line feed).
enter-alt-charset-mode	smacs	as	Starts alternate character set. (P)
enter-blink-mode	blink	mb	Enables blinking.
enter-bold-mode	bold	md	Enables bold (extra bright) mode.
enter-ca-mode	smcup	ti	Begins programs that use cup .
enter-delete-mode	smdc	dm	Starts delete mode.
enter-dim-mode	dim	mh	Enables half-bright mode.
enter-insert-mode	smir	im	Starts insert mode.
enter-protected-mode	prot	mp	Enables protected mode.
enter-reverse-mode	rev	mr	Enables reverse video mode.
enter-secure-mode	invis	mk	Enables blank mode (characters invisible).
enter-standout-mode	smsc	so	Begins standout mode.
enter-underline-mode	smul	us	Starts underscore mode.
erase-chars	ech	ec	Erases #1 characters. (PG)
exit-alt-charset-mode	rmacs	ae	Ends alternate character set. (P)
exit-attribute-mode	sgr0	me	Disables all attributes.
exit-ca-mode	rmcup	te	Ends programs that use cup .

VARIABLE	CAP NAME	I. CODE	DESCRIPTION
exit_delete_mode	rmdc	ed	Ends delete mode.
exit_insert_mode	rmir	ei	Ends insert mode.
exit_standout_mode	rmso	se	Ends stand out mode.
exit_underline_mode	rmul	ue	Ends underscore mode.
flash_screen	flash	vb	Indicates visual bell (may not move cursor).
font_0	font0	f0	Select font 0.
font_1	font1	f1	Select font 1.
font_2	font2	f2	Select font 2.
font_3	font3	f3	Select font 3.
font_4	font4	f4	Select font 4.
font_5	font5	f5	Select font 5.
font_6	font6	f6	Select font 6.
font_7	font7	f7	Select font 7.
form_feed	ff	ff	Ejects page (hard-copy terminal). (P*)
from_status_line	fsl	fs	Returns from status line.
init_lstring	isl	il	Initializes terminal.
init_2string	is2	i2	Initializes terminal.
init_3string	is3	i3	Initializes terminal.
init_file	if	if	Identifies file containing is.
insert_character	ich1	ic	Inserts character. (P)
insert_line	il1	al	Adds new blank line. (P*)
insert_padding	ip	ip	Inserts pad after character inserted. (P*)
key_backspace	kbs	kb	Sent by Backspace key.
key_back_tab	kbt	k0	Sent by backtab key.
key_catab	ktbc	ka	Sent by clear-all-tabs key.
key_clear	kclr	kC	Sent by clear-screen or erase key.
key_ctab	kctab	kt	Sent by clear-tab key.
key_command	kcmd	kc	Command request key.
key_command_pane	kcpn	kW	Command pane key.
key_dc	kdch1	kD	Sent by delete-character key.
key_dl	kdll	kL	Sent by delete-line key.
key_do	kdo	ki	Do request key.
key_down	kcud1	kd	Sent by terminal Cursor Down key.
key_eic	krmir	kM	Sent by rmir or smir in insert mode.
key_end	kend	kw	End key.
key_eol	kel	kE	Sent by clear-to-end-of-line key.
key_eos	ked	kS	Sent by clear-to-end-of-screen key.
key_f0	kf0	k0	Sent by function key F0 .
key_f1	kf1	k1	Sent by function key F1 .

VARIABLE	CAP NAME	I. CODE	DESCRIPTION
key_f2	kf2	k2	Sent by function key F2 .
key_f3	kf3	k3	Sent by function key F3 .
key_f4	kf4	k4	Sent by function key F4 .
key_f5	kf5	k5	Sent by function key F5 .
key_f6	kf6	k6	Sent by function key F6 .
key_f7	kf7	k7	Sent by function key F7 .
key_f8	kf8	k8	Sent by function key F8 .
key_f9	kf9	k9	Sent by function key F9 .
key_f10	kf10	ka	Sent by function key F10 .
key_f11	kf11	k <	Sent by function key F11 .
key_f12	kf12	k >	Sent by function key F12 .
key_help	khlp	kq	Help key.
key_home	khome	kh	Sent by Home key.
key_ic	kich1	kI	Sent by insert character/enter insert mode key.
key_il	kill	kA	Sent by insert line key.
key_left	kcub1	kl	Sent by terminal Cursor Left key.
key_ll	kll	kH	Sent by home-down key.
key_newline	kn1	kn	New-line key.
key_next-pane	knpn	kv	Next-pane key.
key_npage	knp	kN	Sent by next-page key.
key_ppage	kpp	kP	Sent by previous-page key.
key_prev_cmd	kpcmd	kp	Sent by previous-command key.
key_quit	kquit	kQ	Quit key.
key_right	kcuf1	kr	Sent by terminal Cursor Right key.
key_scroll-left	kscl	kz	Scroll left.
key_scroll-right	kscr	kZ	Scroll right.
key_select	ksel	kU	Select key.
key_sf	kind	kF	Sent by scroll-forward/down key.
key_smap-in1	kmpf1	Kv	Input for special mapped key 1.
key_smap-out1	kmpt1	KV	Output for mapped key 1.
key_smap-in2	kmpf2	Kw	Input for special mapped key 2.
key_smap-out2	kmpt2	KW	Output for mapped key 2.
key_smap-in3	kmpf3	Kx	Input for special mapped key 3.
key_smap-out3	kmpt3	KX	Output for mapped key 3.
key_smap-in4	kmpf4	Ky	Input for special mapped key 4.
key_smap-out4	kmpt4	KY	Output for mapped key 4.
key_smap-in5	kmpf5	Kz	Input for special mapped key 5.
key_smap-out5	kmpt5	KZ	Output for mapped key 5.

VARIABLE	CAP NAME	I. CODE	DESCRIPTION
key_sr	kri	kR	Sent by scroll-backward/up key.
key_stab	khts	kT	Sent by set-tab key.
key_tab	ktab	ko	Tab key.
key_up	kcuul	ku	Sent by terminal C ursor U p key.
keypad_local	rmkx	ke	Ends keypad transmit mode.
keypad_xmit	smkx	ks	Puts terminal in keypad transmit mode.
lab_f0	lf0	l0	Labels function key F0 if not F0 .
lab_f1	lf1	l1	Labels function key F1 if not F1 .
lab_f2	lf2	l2	Labels function key F2 if not F2 .
lab_f3	lf3	l3	Labels function key F3 if not F3 .
lab_f4	lf4	l4	Labels function key F4 if not F4 .
lab_f5	lf5	l5	Labels function key F5 if not F5 .
lab_f6	lf6	l6	Labels function key F6 if not F6 .
lab_f7	lf7	l7	Labels function key F7 if not F7 .
lab_f8	lf8	l8	Labels function key F8 if not F8 .
lab_f9	lf9	l9	Labels function key F9 if not F9 .
lab_f10	lf10	la	Labels function key F10 if not F10 .
meta_on	smm	mm	Enables "meta mode" (8th bit).
meta_off	rmm	mo	Disables "meta mode."
newline	nel	nw	Performs new-line function (behaves like CR followed by LF).
pad_char	pad	pc	Pads character (instead of NUL).
parm_dch	dch	DC	Deletes #1 characters. (PG*)
parm_delete_line	dl	DL	Deletes #1 lines. (PG*)
parm_down_cursor	cud	DO	Moves cursor down #1 lines. (PG*)
parm_ich	ich	IC	Inserts #1 blank characters. (PG*)
parm_index	indn	SF	Scrolls forward #1 lines. (PG)
parm_insert_line	il	AL	Adds #1 new blank lines. (PG*)
parm_left_cursor	cub	LE	Moves cursor left #1 spaces. (PG)
parm_right_cursor	cuf	RI	Moves cursor right #1 spaces. (PG*)
parm_rindex	rin	SR	Scrolls backward #1 lines. (PG)
parm_up_cursor	cuu	UP	Moves cursor up #1 lines. (PG*)
pkey_key	pfkey	pk	Programs function key F1 to type string #2.
pkey_local	pfloc	pl	Programs function key F1 to execute string #2.
pkey_xmit	pfx	px	Programs function key F1 to xmit string #2.
print_screen	mc0	ps	Prints contents of the screen.
prtr_off	mc4	pf	Disables the printer.
prtr_on	mc5	po	Enables the printer.

VARIABLE	CAP NAME	I. CODE	DESCRIPTION
repeat_char	rep	rp	Repeats character #1 twice. (PG*)
reset_1string	rs1	r1	Resets terminal to known modes.
reset_2string	rs2	r2	Resets terminal to known modes.
reset_3string	rs3	r3	Resets terminal to known modes.
reset_file	rf	rf	Identifies the file containing reset string.
restore_cursor	rc	rc	Restores cursor to position of last sc.
row_address	vpa	cv	Positions cursor to an absolute vertical position (set row). (PG)
save_cursor	sc	sc	Saves cursor position. (P)
scroll-forward	ind	sf	Scrolls text up. (P)
scroll-reverse	ri	sr	Scrolls text down. (P)
set_attributes	sgr	sa	Defines the video attributes. (PG9)
set-tab	hts	st	Sets a tab in all rows, current column.
set-window	wind	wi	Indicates current window is lines #1 to #2 cols #3 to #4.
tab	ht	ta	Tabs to next 8-space hardware tab stop.
to-status_line	tsl	ts	Moves to status line, column #1.
underline_char	uc	uc	Underscores one character and moves beyond it.
up_half_line	hu	hu	Indicates superscript (reverse 1/2 line-feed).
init_prog	ipro	iP	Locates the program for init .
key-a1	ka1	K1	Specifies upper left of keypad.
key-a3	ka3	K3	Specifies upper right of keypad.
key-b2	kb2	K2	Specifies center of keypad.
key-c1	kc1	K4	Specifies lower left of keypad.
key-c3	kc3	K5	Specifies lower right of keypad.
prtr-non	mc5p	pO	Enables the printer for #1 bytes.

Terminal capabilities have names. For instance, the fact that a terminal has automatic margins (such as, an automatic new-line when the end of a line is reached) is indicated by the capability **am**. Hence the description of the terminal includes **am**. Numeric capabilities are followed by the # (pound) character and then the value. Thus the **cols#80** capability, which indicates the number of columns the terminal has, gives the value 80 for the terminal.

Finally, string-valued capabilities, such as **el** (clear to end of line sequence) are given by the 2-character code, an = (equal sign), and then a string ending at the following , (comma). A delay in milliseconds may appear anywhere in a string capability, enclosed between a \$ < and a > as in **el=\EK\$<3>**, and padding characters are supplied by **tputs** to provide this delay. The delay can be either a number, such as 20, or a number followed by an * (asterisk), such as 3*. An asterisk indicates that the padding required is proportional to the number of lines affected by the operation, and the amount given is the per-affected-unit padding required. (In the case of insert character, the factor is still the

number of **lines** affected. This is always 1, unless the terminal has **xenl** and the software uses it.) When an asterisk is specified, it is sometimes useful to give a delay of the form **a.b**, such as, 3.5, to specify a delay per unit to tenths of milliseconds. (Only one decimal place is allowed.)

A number of escape sequences are provided in the string-valued capabilities for easy encoding of characters there. Both **\E** and **\e** map to an **Escape** character, **^x** maps to a **Ctrl-x** for any appropriate x, and the sequences **\n**, **\l**, **\r**, **\t**, **\b**, **\f**, **\s** give a new-line, line feed, return, tab, backspace, form-feed, and space. Other escapes include **\^** (backslash caret) for a **^** (caret), **** (backslash backslash) for a **** (backslash), **\,** (backslash comma) for a **,** (comma), **\:** (backslash colon) for a **:** (colon), and **\0** (backslash) for the null character. (**\0** will produce **\200**, which does not terminate a string but behaves as a null character on most terminals.) Finally, characters can be given as 3 octal digits after a **** (backslash).

Sometimes, individual capabilities must be commented out. To do this, put a period before the capability name.

Preparing Descriptions

An effective way to prepare a terminal description is to imitate the description of a similar terminal in the **terminfo** file and add to the description gradually, using partial descriptions with **vi** to check that they are correct. Be aware that a very unusual terminal can expose deficiencies in the ability of this file to describe it or bugs in **vi**. To test a new terminal description, set the environment variable **TERMINFO** to a path name of a directory containing the compiled description you are working on and programs will look there rather than in **/usr/lib/terminfo**. A test to get the correct padding (if not known) is to edit the **/etc/passwd** file at 9600 baud, delete about 16 lines from the middle of the screen, then hit the **U** key several times quickly. If the terminal fails to display the result properly, more padding is usually needed. A similar test can be used for insert character.

Basic Capabilities

The following describe basic terminal capabilities:

am	Indicates that the cursor moves to the beginning of the next line when it reaches the right margin. This capability also indicates whether the cursor can move beyond the bottom right corner of the screen.
bel	Produces an audible signal (such as a bell or a beep).
bw	Indicates that a backspace from the left edge of the terminal moves the cursor to the last column of the previous row.
clear	Clears the screen leaving the cursor in the home position.
cols	Specifies the number of columns on each line for the terminal.
cr	Moves the cursor to the left edge of the current row. This code is usually carriage return (Ctrl-M).

- cub1** Moves the cursor one space to the left, such as backspace.
- cuf1, cuu1, and cud1** Moves the cursor to the right, up, and down, respectively.
- hc** Specifies a printing terminal. The **os** capability should also be specified.
- lines** Specifies the number of lines on a cathode ray tube (CRT) terminal.
- os** Indicates that when a character is displayed or printed in a position already occupied by another character, the terminal overstrikes the existing character, rather than replacing it with the new character. **os** applies to storage scope, printing, and APL terminals.

The **terminfo** initialization subroutine, **setupterm**, calls **termdef** to determine the number of lines and columns on the display. If **termdef** cannot supply this information, then **setupterm** uses the **lines** and **cols** values in the data base.

A point to note here is that the local cursor motions encoded in **terminfo** are undefined at the left and top edges of a CRT terminal. Programs should never attempt to backspace around the left edge, unless **bw** is given, and never attempt to go up locally off the top. In order to scroll text up, a program should go to the bottom left corner of the screen and send the **ind** (index) string.

To scroll text down, a program goes to the top left corner of the screen and sends the **ri** (reverse index) string. The strings **ind** and **ri** are undefined when not on their respective corners of the screen.

The **am** capability tells whether the cursor sticks at the right edge of the screen when text is output, but this does not necessarily apply to a **cuf1** from the last column. The only local motion that is defined from the left edge is if **bw** is given, then a **cub1** from the left edge will move to the right edge of the previous row. If **bw** is not given, the effect is undefined. This is useful for drawing a box around the edge of the screen, for example. If the terminal has switch-selectable automatic margins, the **terminfo** file usually assumes that it is on by specifying **am**. If the terminal has a command that moves to the first column of the next line, that command can be given as **nel** (new-line). It does not matter if the command clears the remainder of the current line, so if the terminal has no **cr** and **lf**, it may still be possible to craft a working **nel** out of one or both of them.

These capabilities suffice to describe printing terminals and simple CRT terminals. Thus, the Model 33 Teletype is described as:

```
33 | tty33 | tty | Model 33 Teletype,  
    bel=^G, cols#72, cr=^M, cudl=^J, hc, ind=^J, os,
```

And another terminal is described as:

```
xxxx | x | xxxxxxxxx,  
    am, bel=^G, clear=^Z, cols#80, cr=^M, cubl=^H, cudl=^J,  
    ind=^J, lines#24,
```

Parameterized Strings

Cursor addressing and other strings requiring parameters in the terminal are described by a parameterized string capability, with escapes similar to **printf %x** in it. For example, to address the cursor, the **cup** capability is given using two parameters: the row and column to address to. (Rows and columns are numbered starting with 0 and refer to the physical screen visible to the user, not to any unseen memory.) If the terminal has memory relative cursor addressing, that can be indicated by **mrcup**.

The parameterized capabilities and their descriptions are:

cubl Backspaces the cursor one space.

cup Addresses the cursor using two parameters: the row and column to address. Rows and columns are numbered starting with 0 and refer to the physical screen visible to the user, not to memory.

cuu1 Moves the cursor up one line on the screen.

hpa and **vpa**

Indicates the cursor has row or column absolute cursor addressing, horizontal position absolute (**hpa**) and vertical position absolute (**vpa**).

Sometimes the **hpa** and **vpa** capabilities are shorter than the more general two-parameter sequence and can be used in preference to **cup**. If there are parameterized local motions (such as, move **n** spaces to the right) these can be given as **cud**, **cub**, **cuf**, and **cuu** with a single parameter indicating how many spaces to move. These are primarily useful if the terminal does not have **cup**.

indn and **rin**

Scrolls text. These are parameterized versions of the basic capabilities **ind** and **ri**. *n* is the number of lines.

mrcup Indicates the terminal has memory-relative cursor addressing.

The parameter mechanism uses a stack and special **%** codes to manipulate it. Typically a sequence pushes one of the parameters onto the stack and then prints it in some format. Often more complex operations are necessary.

The % encodings have the following meanings:

%%	Outputs a % (percent) character.
%d	Print pop() as in printf (numeric string from stack).
%2d	Print pop() like %2d (minimum 2 digits output from stack).
%3d	Print pop() like %3d (minimum 3 digits output from stack).
%02d	Prints as in printf (2 digits output).
%03d	Prints as in printf (3 digits output).
%c	Print pop() gives %c (character output from stack).
%s	Print pop() gives %s (string output from stack).
%p[i]	Pushes the <i>i</i> th parameter onto stack.
%P[a-z]	Sets variable [a-z] to pop() (variable output from stack).
%g[a-z]	Gets variable [a-z] and pushes it onto the stack.
%'c'	Character constant <i>c</i> .
%{nn}	Integer constant <i>nn</i> .
%+ %- %* %/ %m	Arithmetic (%m is modulus): push(pop() <i>operation</i> pop())
%& % %^	Bit operations: push(pop() <i>operation</i> pop())
%= %> %<	Logical operations: push(pop() <i>operation</i> pop()).
%! %~	Unary operations push(<i>operation</i> pop())
%i	Add 1 to first two parameters (for ANSI terminals).

/? expr %t thenpart %e elsepart %;

If-then-else. The **%e elsepart** is optional. You can make an else-if construct as with Algol 68:

/? c₁ %t b₁ %e c₂ %t b₂ %e c₃ %t b₃ %e b₄ %;

In this example, *c_i* denote conditions, and *b_i* denote bodies.

Binary operations are in postfix form with the operands in the usual order. That is, to get *x - 5* one would use **%gx%{5}%-**.

Consider a terminal, which, to get to row 3 and column 12, needs to be sent **\E&a12c03Y** padded for 6 milliseconds. Note that the order of the rows and columns is inverted here, and that the row and column are printed as two digits. Thus its **cup** capability is **cup=6\E&a%p2%2dc%p1%2dY**.

Some terminals need the current row and column sent preceded by a **^T** with the row and column simply encoded in binary, **cup=^Tp1%c%p2%c**. Terminals which use **%c** need to be able to backspace the cursor (**cu_b1**), and to move the cursor up one line on the screen (**cu_u1**). This is necessary because it is not always safe to transmit **\n**, **^D**, and **\r**, as the system may change or discard them. (The library routines dealing with **terminfo** set

terminal modes so that tabs are not expanded by the operating system; thus `\t` is safe to send.)

A final example is a terminal that uses row and column offset by a blank character, thus `cup=\E=%p1%' '%+%c%p2%' '%+%c`. After sending `\E=`, this pushes the first parameter, pushes the ASCII value for a space (32), adds them (pushing the sum on the stack in place of the two previous values) and outputs that value as a character. Then the same is done for the second parameter. More complex arithmetic is possible using the stack.

Cursor Motions

If the terminal has a fast way to home the cursor (to very upper left corner of screen) then this can be given as **home**. Similarly a fast way of getting to the lower left-hand corner can be given as **ll**; this may involve going up with **cuu1** from the home position, but a program should never do this itself (unless **ll** does) because it can make no assumption about the effect of moving up from the home position. Note that the home position is the same as addressing (0,0) to the top left corner of the screen, not of memory. (Thus, the `\EH` sequence on some terminals cannot be used for **home**.)

Area Clears

The following areas are used to clear large areas of the terminal:

- ed** Clears from the current position to the end of the display. This is defined only from the first column of a line. (Thus, it can be simulated by a request to delete a large number of lines, if a true **ed** is not available.)
- el** Clears from the current cursor position to the end of the line without moving the cursor.

Insert/Delete Line

The following describes the insert and delete line capabilities:

- csr** Indicates the terminal has a scrolling region that can be set. This capability takes two parameters: the top and bottom lines of the scrolling region.
- da** Indicates the terminal can retain display memory above what is visible.
- db** Indicates the display memory can be retained below what is visible.
- dl1** Indicates the line the cursor is on can be deleted. This done only from the first position on the line to be deleted. Additionally, the **dl** capability takes a single parameter indicating the number of lines to be deleted.
- il1** Creates a new blank line before the line where the cursor is currently located and scrolls the rest of the screen down. This is done only from the first position of a line. The cursor then appears on the newly blank line. Additionally, the **il** capability can take a single parameter indicating the number of lines to insert.

- rc** Restores the cursor. When used after the **csr** capability, it gives an effect similar to delete line.
- sc** Saves the cursor. When used after the **csr** capability, it gives an effect similar to insert line.
- wind** Indicates the terminal has the ability to define a window as part of memory. This a parameterized string with four parameters: the starting and ending lines in memory and the starting and ending columns in memory, in that order.

Insert/Delete Character

Generally, there are two kinds of programmable terminals, with respect to insert/delete character operations, which can be described using the **terminfo** file. The most common insert/delete character operations affect only the characters on the current line and shift characters to the right and off the line. Other terminals make a distinction between typed and untyped blanks on the screen, shifting data displayed to insert or delete at a position on the screen occupied by an untyped blank, which is either eliminated or expanded to two untyped blanks. Clearing the screen and then typing text separated by cursor motions differentiates between the terminal types. You can determine the kind of terminal you have by doing the following:

1. Type **abc def** using local cursor movements, not spaces, between the **abc** and the **def**.
2. Position the cursor before the **abc** and place the terminal in insert mode. If typing characters causes the characters on the line to the right of the cursor to shift and exit the right side of the display, the terminal does not distinguish between blanks and untyped positions. If the **abc** moves to positions to the immediate left of the **def** and the characters move to the right on the line, around the end, and to the next line, the terminal does distinguish between blanks and untyped positions. This is described by the **in** capability, which signifies insert null.

While these are two logically separate attributes (one line vs. multiline insert mode, and special treatment of untyped spaces) there are no known terminals whose insert mode cannot be described with the single attribute.

The **terminfo** file can describe both terminals having an insert mode and terminals that send a simple sequence to open a blank position on the current line. The following are used to describe insert or delete character capabilities:

- dch1** Deletes a single character. **dch** with one parameter, **n** deletes **n** characters.
- ech** Erases **n** characters (equivalent to typing **n** blanks without moving the cursor) with one parameter.
- ich1** Precedes the character to be inserted. Most terminals with an insert mode do not use this. Terminals that send a sequence to open a screen position should give it. (If the terminal has both, insert mode is usually preferable to **ich1**. Do

	not give both unless the terminal actually requires both to be used in combination.)
ip	Indicates post padding needed. This is given as a number of milliseconds. Any other sequence that may need to be sent after inserting a single character can be given in this capability.
mir	Allows cursor motion while in insert mode. It is sometimes necessary to move the cursor while in insert mode to delete characters on the same line. Some terminals may not have this capability due to their handling of insert mode.
rmde	Exits delete mode.
rmir	Ends insert mode.
smde	Enters delete mode.
smir	Begins insert mode.

Note that if your terminal needs both to be placed into an insert mode and a special code to precede each inserted character, then both **smir/rmir** and **ich1** can be given, and both will be used. The **ich** capability, with one parameter, **n**, will repeat the effects of **ich1 n** times.

Highlighting, Underlining, and Visual Bells

If your terminal has one or more kinds of display attributes such as highlighting, underlining, and visual bells, these can be presented in a number of ways. Highlighting, such as **standout mode**, presents a good, high contrast, easy-on-the-eyes format to add emphasis to error messages, and other attention getters. Underlining is another method to focus attention to a particular portion of the terminal. Visual bells include methods such as flashing the screen. The following capabilities describe highlighting, underlining, and visual bells for a terminal:

blink	Indicates terminal has blink highlighting mode.
bold	Indicates terminal has extra bright highlighting mode.
civis	Causes the cursor to be invisible.
cnorm	Causes the cursor to display normal. This capability reverses the effects of the civis and cvvis capabilities.
cvvis	Causes the cursor to be more visible than normal when it is not on the bottom line.
dim	Indicates the terminal has half-bright highlighting modes.
eo	Indicates blanks erase overstrikes.
flash	Indicates the terminal has a way of flashing the screen (a bell replacement) for errors without moving the cursor.

invis	Indicates the terminal has blanking or invisible text highlighting modes.
msgr	Indicates it is safe to move the cursor while in standout mode. Otherwise, programs using standout mode should exit standout mode before moving the cursor or sending a new-line. Some terminals automatically leave standout mode when they move to a new line or the cursor is addressed.
prot	Indicates the terminal has protected highlighting mode.
rev	Indicates the terminal has reverse video mode.
rmso	Exits standout mode.
rmul	Ends underlining.
sgr	Sets attributes. sgr0 turns off all attributes. Otherwise, if the terminal allows a sequence to set arbitrary combinations of modes, sgr takes nine parameters. Each parameter is either 0 or 1, as the corresponding attribute is on or off. The nine parameters are in this order: standout, underline, reverse, blink, dim, bold, blank, protect, and alternate character set. (sgr can only support those modes for which separate attributes exist on a particular terminal.)
smcup and rmcup	Indicate that the terminal needs to be in a special mode when running a program that uses any of the highlighting, underlining or visual bell capabilities. smcup enters this mode, while rmcup exits this mode. This need arises, for example, from terminals with more than one page of memory. If the terminal has only memory relative cursor addressing, and not screen relative cursor addressing, a screen-sized window must be fixed into the terminal for cursor addressing to work properly. This is also used where smcup sets the command character to be used by the terminfo file.
smso	Enters standout mode.
smul	Begins underlining.
uc	Underlines the current character and moves the cursor one space to the right.
ul	Indicates the terminal correctly generates underlined characters (with no special codes needed) even though it does not overstrike.
xmc	Indicates the number of blanks left if the capability to enter or exit standout mode leaves blank spaces on the screen.

Keypad

If the terminal has a keypad that transmits codes when the keys are pressed, this information can be given. Note that it is not possible to handle terminals where the keypad only works in local mode. If the keypad can be set to transmit or not transmit, give these codes as **smkx** and **rmkx**. Otherwise the keypad is assumed to always transmit. The codes sent by the Left Arrow, Right Arrow, Up Arrow, Down Arrow, and **Home** keys can be given as **kcub1**, **kcuf1**, **kcuu1**, **kcud1**, and **khome**, respectively. If there are function keys such as **F0**, **F1**, . . . , **F10**, the codes they send can be given as **kf0**, **kf1**, . . . , **kf10**. If these keys have labels other than the default **F0** through **F10**, the labels can be given as **lf0**, **lf1**, . . . , **lf10**. The codes transmitted by certain other special keys can be given as:

kbs	Indicates the Backspace key.
kclr	Indicates the clear screen or erase key.
kctab	Indicates clear the tab stop in this column.
kdch1	Indicates the delete character key.
kdl1	Indicates the delete line key.
ked	Indicates clear to end of screen.
kel	Indicates clear to end of line.
khts	Indicates set a tab stop in this column.
kich1	Indicates insert character or enter insert mode.
kill	Indicates insert line.
kind	Indicates scroll forward and/or down.
kll	Indicates home down key (home is the lower left corner of the display, in this instance).
kmir	Indicates exit insert mode.
knp	Indicates next page.
kpp	Indicates previous page.
ktbc	Indicates the clear all tabs key.
ri	Indicates scroll backward and/or up.

In addition, if the keypad has a 3-by-3 array of keys including the four arrow keys, the other five keys can be given as **ka1**, **ka3**, **kb2**, **kc1**, and **kc3**. These keys are useful when the effects of a 3-by-3 directional pad are needed.

Tabs and Initialization

If the terminal has hardware tabs, the command to advance to the next tab stop can be given as **ht** (usually **Ctrl-I**). A “backtab” command which moves left toward the previous tab stop can be given as **cbt**. By convention, if the terminal modes indicate that tabs are being expanded by the operating system rather than being sent to the terminal, programs should not use **ht** or **cbt** even if they are present, since the user may not have the tab stops properly set. If the terminal has hardware tabs that are initially set every **n** spaces when the terminal is powered up, the numeric parameter **it** is given, showing the number of spaces the tabs are set to. This is normally used by the **tset** command to determine whether to set the mode for hardware tab expansion, and whether to set the tab stops. If the terminal has tab stops that can be saved in nonvolatile memory, the **terminfo** description can assume that they are properly set.

Other capabilities include **is1**, **is2**, and **is3**, initialization strings for the terminal, **iprog**, the path name of a program to be run to initialize the terminal, and **if**, the name of a file containing long initialization strings. These strings are expected to set the terminal into modes consistent with the rest of the **terminfo** description. They are normally sent to the terminal, by the **tset** program, each time the user logs in. They are printed in the following order: **is1**, **is2**, setting tabs using **tbc** and **hts**; **if**; running the program **iprog**; and finally **is3**. Most initialization is done with **is2**. Special terminal modes can be set up without duplicating strings by putting the common sequences in **is2** and special cases in **is1** and **is3**. A pair of sequences that does a harder reset from a totally unknown state can be analogously given as **rs1**, **rs2**, **rf**, and **rs3**, analogous to **is2** and **if**. These strings are output by the **reset** program, which is used when the terminal starts behaving strangely, or not responding at all. Commands are normally placed in **rs2** and **rf** only if they produce annoying effects on the screen and are not necessary when logging in. For example, the command to set the terminal into 80-column mode would normally be part of **is2**, but it causes an annoying screen behavior and is not normally needed since the terminal is usually already in 80-column mode.

If there are commands to set and clear tab stops, they can be given as **tbc** (clear all tab stops) and **hts** (set a tab stop in the current column of every row). If a more complex sequence is needed to set the tabs than can be described by this, the sequence can be placed in **is2** or **if**.

Certain capabilities control padding in the terminal driver. These are primarily needed by hard copy terminals, and are used by the **tset** program to set terminal modes appropriately. Delays embedded in the capabilities **cr**, **ind**, **cub1**, **ff**, and **tab** cause the appropriate delay bits to be set in the terminal driver. If **pb** (padding baud rate) is given, these values can be ignored at baud rates below the value of **pb**.

Miscellaneous Strings

If the terminal requires other than a null (zero) character as a pad, then this can be given as **pad**. Only the first character of the **pad** string is used.

If the terminal has an extra “status line” that is not normally used by software, this fact can be indicated. If the status line is viewed as an extra line below the bottom line, which you can use the cursor to address normally, the capability **hs** should be given. Special strings to go to the beginning of the status line and to return from the status line can be given as **tsl** and **fsl**. (**fsl** must leave the cursor position in the same place it was before **tsl**. If necessary, the **sc** and **rc** strings can be included in **tsl** and **fsl** to get this effect.) The parameter **tsl** takes one parameter, which is the column number of the status line the cursor is to be moved to. If escape sequences and other special commands, such as tab, work while in the status line, the flag **eslok** can be given. A string that turns off the status line (or otherwise erases its contents) should be given as **dsl**. If the terminal has commands to save and restore the position of the cursor, give them as **sc** and **rc**. The status line is normally assumed to be the same width as the rest of the screen, such as **cols**. If the status line is a different width (possibly because the terminal does not allow an entire line to be loaded) the width, in columns, can be indicated with the numeric parameter **wsl**.

If the terminal can move up or down half a line, this can be indicated with **hu** (half-line up) and **hd** (half-line down). This is primarily useful for superscripts and subscripts on hard copy terminals. If a hard copy terminal can eject to the next page (form feed), give this as **ff** (usually **Ctrl-L**).

To save time transmitting a large number of identical characters, you can repeat a given character a given number of times by using the parameters in the string **rep**. The first parameter is the character to be repeated and the second is the number of times to repeat it. Thus, `tparam(repeat_char, 'x', 10)` is the same as `XXXXXXXXXX`.

If the terminal has a meta key that acts as a shift key, setting the eighth bit of any character transmitted, this fact can be indicated with **km**. Otherwise, software will assume that the eighth bit is parity and it will usually be cleared. If strings exist to turn this “meta mode” on and off, they can be given as **smm** and **rmm**.

If the terminal has more lines of memory than will fit on the screen at once, the number of lines of memory can be indicated with **lm**. A value of **lm#0** indicates that the number of lines is not fixed, but that there is still more memory than fits on the screen.

Media copy strings that control an auxiliary printer connected to the terminal can be given in the following ways: **mc0** prints the contents of the screen, **mc4** turns off the printer, and **mc5** turns on the printer. When the printer is on, all text sent to the terminal is sent to the printer. It is undefined whether the text is also displayed on the terminal screen when the printer is on. A variation **mc5p** takes one parameter, and leaves the printer on for as many characters as the value of the parameter, then turns the printer off. The parameter should not exceed 255. All text, including **mc4**, is transparently passed to the printer while an **mc5p** is in effect.

Strings to program function keys can be given as **pfkey**, **pfloc**, and **pfx**. Each of these strings takes two parameters: the function key number to program (from 0 to 10) and the string to program it with. Function key numbers out of this range can program undefined keys in a terminal-dependent manner. The difference between the capabilities is that **pfkey** causes pressing the given key to be the same as the user typing the given string; **pfloc** causes the string to be executed by the terminal in local mode; and **pfx** causes the string to be transmitted to the computer.

Indicating Terminal Problems

Terminals that do not allow ~ (tilde) characters to be displayed should indicate **hz**.

Terminals that ignore a line feed character immediately after an **am** wrap should indicate **xenl**.

If **el** is required to get rid of standout (instead of merely writing normal text on top of it), **xhp** should be given.

Terminals for which tabs turn all characters moved to blanks should indicate **xt** (destructive tabs). This capability is interpreted to mean that it is not possible to position the cursor on top of the pads inserted for standout mode. Instead, it is necessary to erase standout mode using delete and insert line.

The terminal that is unable to correctly transmit the **Esc** (escape) or **Ctrl-C** characters has **xsb**, indicating that the **F1** key is used for **Esc** and **F2** for **Ctrl-C**.

Other specific terminal problems can be corrected by adding more capabilities of the form **xx**.

Similar Terminals

If two terminals are very similar, one can be defined as being just like the other with certain exceptions. The string capability **use** can be given with the name of the similar terminal. The capabilities given before **use** override those in the terminal type invoked by **use**. A capability can be canceled by placing **xx@** to the left of the capability definition, where **xx** is the capability. For example, the entry:

```
term-nl, smkx@, rmkx@, use=term,
```

defines a terminal that does not have the **smkx** or **rmkx** capabilities, and hence does not turn on the function key labels when in visual mode. This is useful for different modes for a terminal, or for different user preferences.

Data Base File Names

Compiled **terminfo** descriptions are placed in subdirectories under **/usr/lib/terminfo** in order to avoid performing linear searches through a single directory containing all of the **terminfo** description files. A given description file is stored in **/usr/lib/terminfo/c/name**, where *name* is the name of the terminal, and *c* is the first letter of the terminal name. For example, the compiled description for the terminal **term4-nl** can be found in the file **/usr/lib/terminfo/t/term4-nl**. You can create synonyms for the same terminal by making multiple links to the same compiled file. (See the **ln** command in *AIX Operating System Commands Reference* to create multiple links to a file.)

Example

The following entry, which describes a terminal, is among the entries in the **terminfo** file.

```
hft|High Function Terminal,
  cr=^M, cud1=\E[B, ind=\E[S, bel=^G, ill=\E[L, am, cub1=^H, ed=\E[J,
  el=\E[K, clear=\E[H\E[J, cup=\E[%ip1%d;%p2%dH, cols#80, lines#25,
  dch1=\E[P, dll=\E[M, home=\E[H,
  ich=\E[%p1%d@, ich1=\E[@, smir=\E[6, rmir=\E6,
  bold=\E[lm, rev=\E[7m, blink=\E[5m, invis=\E[8m, sgr0=\E[0m,
  sgr=\E[?%p1%t7;%;%?%p2%t4;%;%?%p3%t7;%;%?%p4%t5;%;%?%p6%t1;%;m,
  kcuu1=\E[A, kcud1=\E[B, kcub1=\E[D,
  kcufl=\E[C, khome=\E[H, kbs=^H,
  cufl=\E[C, ht=^I, cuu1=\E[A, xon,
  rmul=\E[m, smul=\E[4m, rmso=\E[m, smso=\E[7m,
  kpp=\E[150q, knp=\E[154q,
  kf1=\E[001q, kf2=\E[002q, kf3=\E[003q, kf4=\E[004q,
  kf5=\E[005q, kf6=\E[006q, kf7=\E[007q, kf8=\E[008q,
  kf9=\E[009q, kf10=\E[010q,
  bw, eo, it#8, ms,
  ch=\E%i%p1%dG, ech=\E[%p15dx,
  kdch1=\E[P, kind=\E[151q, kich1=\E[139q, krmir \E[41,
  kn=^M, ko=^I, ktab=\E[Z, kri=\E[155q,
  cub=\E[%p1%dD, cuf=\E[%p1%dC, indn=\E[%p1dS, rin=\E[%p1dT,
  ri=\E[T, cuu=\E[%p1dA,
  box1=\332\304\277\263\331\300\302\264\301\303\305,
  box2=\311\315\273\272\274\310\313\271\312\314\316,
  batt2=md,
  colf0=\E[30m, colf1=\E[31m, colf2=\E[32m, colf3=\E[33m,
```

colf4=\E[34m,	colf5=\E[35m,	colf6=\E[36m,	colf7=\E[37m,
colb0=\E[40m,	colb1=\E[41m,	colb2=\E[42m,	colb3=\E[43m,
colb4=\E[44m,	colb5=\E[45m,	colb6=\E[46m,	colb7=\E[47m,

File

`/usr/lib/terminfo/?/*` Compiled terminal capability data base.

Related Information

In this book: “curses” on page 2-129, “Terminfo Level Subroutines” on page 2-135, “extended curses library” on page 2-238, “printf, fprintf, sprintf, NLprintf, NLfprintf, NLsprintf” on page 2-553, “termdef” on page 2-814, and “TERM” on page 4-76.

The **display** and **tic** commands in *AIX Operating System Commands Reference*.

trdconf

trdconf

Purpose

Defines the configuration file for Token-Ring diagnostics.

Description

The **/etc/trdconf** file defines the attributes of the log files produced by the Token-Ring diagnostics.

The **trdconf** file consists of two stanzas. The **trdlog** stanza describes the Token-Ring diagnostics log file, and the **MAClog** stanza describes the medium access control (MAC) frame log file.

The following attributes are valid:

- | | |
|-----------------------------|--|
| file = <i>file</i> | Specifies the file into which the Token-Ring diagnostics writes the specified information. |
| size = <i>blocks</i> | Specifies the maximum size, in 1024-byte blocks, to which the file will grow. |

Example

The **trdconf** file provided with the system contains the following:

```
trdlog:
  file = /usr/adm/ras/trdfile
  size = 50
```

```
MAClog:
  file = /usr/adm/ras/Mfile
  size = 100
```


File

/etc/trdconf

Related Information

In this book: “attributes” on page 3-20, “trdfile” on page 3-244, and “Mfile” on page 3-136.

The discussion of “Token-Ring Diagnostics” in *Managing the AIX Operating System*.

The **trdiag** command in *AIX Operating System Commands Reference*.

trdfile

trdfile

Purpose

Contains the Token-Ring diagnostics log.

Description

When Token-Ring diagnostics is running, warning and error messages that are sent as a result of error analysis and informational messages indicating changes in the Token-Ring diagnostics parameters or in the Token-Ring status are logged in a file. The `/etc/trdconf` file specifies the default file where Token-Ring messages are logged. The installed `/etc/trdconf` file contains the default Token-Ring diagnostics log file name of `/usr/adm/ras/trdfile`.

Token-Ring diagnostics messages are placed in one of two log files. The Token-Ring diagnostics creates the names of the two files by adding to the end of the file name found in `/etc/trdconf` the number from the Token-Ring device name plus `.0` and `.1`. For example, for device `token0` the log files are named `trdfile0.0` and `trdfile0.1`.

Messages are first added to `file.0` until it reaches the maximum allowable length as specified in `/etc/trdconf`. At that point, Token-Ring diagnostics closes `file.0`, renames `file.0` to `file.1`, and opens a new `file.0`. Thus, the newest records are always in `file.0`.

For information on messages placed in the Token-Ring diagnostics log file, see the description of messages from the data messages area in "Token-Ring Diagnostics" in *Managing the AIX Operating System*. Information in the log file is identical to that described, with one exception; a 3-byte header is appended to the front of each line of data. For messages that do not contain variable data, the message line is preceded by `10b`. For messages that have variable data, the message is preceded by `11b` and the variable data is preceded by `2bb`.

Files

`/usr/adm/ras/trdfile`
`/etc/trdconf`

Related Information

In this book: “trdconf” on page 3-242.

“Token-Ring Diagnostics” in *Managing the AIX Operating System*.

The **trdiag** command in *AIX Operating System Commands Reference*.

utmp, wtmp, .ilog

utmp, wtmp, .ilog

Purpose

Contains user and accounting information.

Synopsis

```
#include <utmp.h>
```

Description

When a user logs in successfully, the **login** program writes entries in **/etc/utmp**, the record of users logged into the system, and in **/usr/adm/wtmp** (if it exists), for use in accounting. On invalid login attempts (due to an incorrect login name or password), **login** makes entries in the **/etc/.ilog** file. When you log in as user **root** or **su** and the **/etc/.ilog** file is not empty, you see a message advising you to check the **/etc/.ilog** file for a record of unsuccessful login attempts.

The records in these files follow the **utmp** structure, which is defined in the **utmp.h** header file:

```
#define UTMP_FILE   "/etc/utmp"
#define WTMP_FILE   "/usr/adm/wtmp"
#define ILOG_FILE   "/etc/.ilog"

#define ut_name      ut_user
#define ut_id        ut_line

struct utmp {
    char ut_user[8];           /* User login name */
    char ut_line[12];          /* device name (console, lnxx) */
    short ut_pid;              /* process id */
    short ut_type;             /* type of entry */
    struct exit_status {
        short e_termination; /* Process termination status */
        short e_exit;         /* Process exit status */
    } ut_exit;                /* The exit status of a process */
    /* marked as DEAD_PROCESS. */
};
```

```
        time_t  ut_time;      /* time entry was made */
};

/* Definitions for ut_type */

#define EMPTY          0
#define RUN_LVL        1
#define BOOT_TIME      2
#define OLD_TIME        3
#define NEW_TIME        4
#define INIT_PROCESS    5    /* Process spawned by "init" */
#define LOGIN_PROCESS   6    /* A "getty" process waiting for login */
#define USER_PROCESS    7    /* A user process */
#define DEAD_PROCESS    8
#define ACCOUNTING      9
#define UTMXTYPE ACCOUNTING /* Largest legal value of ut_type */

/* Special strings or formats used in the "ut_line" field when */
/* accounting for something other than a process. */
/* No string for the ut_line field can be more than 11 chars + */
/* a NULL in length. */

#define RUNLVL_MSG      "run-level ?"
#define BOOT_MSG        "system boot"
#define OTIME_MSG       "old time"
#define NTIME_MSG       "new time"
```

Files

/etc/utmp	Record of users logged in to the system.
/usr/adm/wtmp	Accounting information
/etc.ilog	Record of invalid logins.

Related Information

The **login**, **who**, and **write** commands in *AIX Operating System Commands Reference*.

vfs

Purpose

Describes the virtual file systems (vfs) that are installed on the system.

Description

The **vfs** file describes the virtual file systems (vfs) that are installed on the system. The name, type number, and file system helper program number are among the types of information listed in the file. Commands, such as **mount**, **fsck** (file system check), and **mkfs** (make file system), use this information.

The **vfs** file is a flat file, with one record per line. Each line is terminated by a new-line escape character. There are three types of lines in the **vfs** file:

- Comments
- General control
- Entries.

The comments, which are designated by a # (pound) character are in text for explanatory purposes. The general control lines, which are designated by a % (percent) character, configure the actions of the **mount** and **umount** commands. For example, a line that appears as **%defaultvfs** indicates the default local vfs for use if no vfs is specified on the **mount** command or in the **/etc/filesystems** file. The entry is the name of the vfs as it appears in the file. If a second entry appears on the same line, it is taken to be the default remote vfs.

The following is an example **vfs** file:

```
aix 0    none
ds 1     /etc/dsmnthelp
nfs 2     /etc/nfsmnthelp
%defaultvfs aix nfs
```

The vfs entries take the following form:

name	Canonical name of this type of vfs.
type	Decimal representation of the vfs type number for the virtual file system.
mnt-helper	Full path name of the mount helper program of this vfs. If a mount helper is not required, the entry should appear as none .

vfs

fs_helper	Full pathname of the file system helper program of this vfs. If a file system helper is not required, the entry should appear as none.
vfs_info	Other information that the vfs needs. The information must be in printable ASCII characters. This is an optional entry.
< new-line >	An ASCII new-line character that marks the end of each record in the file.

Files

/etc/vfs
/etc/filesystems

Related Information

In this book, see “mount” on page 2-460 and “filesystems” on page 3-74.

The **mount** command in *AIX Operating System Commands Reference*.

YP-MAP-X-LATE

Purpose

Provides translated names for Yellow Pages maps.

Description

AIX file names can contain a maximum of 14 characters, but the file names of Yellow Pages data base maps can contain up to 256 characters which includes the file extensions **.pag** and **.dir**. For example, the YP data base file names `passwd.byname.pag` and `passwd.byname.dir` each contain seventeen characters. The **YP-MAP-X-LATE** file provides the translations from the Yellow Pages data base file (or long) names to the AIX (or short) names. The users and the network need not be aware of the translation requirement. Only the server side requires the use of the translated names. When the Yellow Pages processes **ypserv**, **yppush**, and **ypxfr** look up information in the YP data bases for YP clients, they invoke an internal process that consults the **YP-MAP-X-LATE** file for the map's translated file name on their behalf.

Users with superuser authority can edit the **YP-MAP-X-LATE** file to add file name translations for YP maps that are created in their network. See *Managing the AIX Operating System* for information on creating additional YP maps.

The **YP-MAP-X-LATE** file consists of lines that contain the original YP map name followed by the AIX file name. The names are separated by white space. The AIX file names can contain a maximum of ten (10) characters so that when the dbm file extensions **.pag** and **.dir** are added the total characters in the short file name does not exceed 14 characters.

Example

The following is an example of the **YP-MAP-X-LATE** file included with the IBM AIX/RT Network File System:

<code>ethers.byaddr</code>	<code>ethers.adr</code>
<code>ethers.byname</code>	<code>ethers.nam</code>
<code>group.bygid</code>	<code>group.gid</code>

YP_MAP_X_LATE

group.byname	group.nam
aliases	mail.alias
netgroup	netgroup
netgroup.byhost	netgrp.hst
netgroup.byuser	netgrp.usr
networks.byname	netwks.nam
networks.byaddr	netwks.adr
passwd.byname	passwd.nam
passwd.byuid	passwd.uid
protocols.byname	proto.nam
protocols.bynumber	proto.no
rpc.bynumber	rpc.no
services.byname	srvc.s.nam
ypservers	ypservers

Files

/etc/yp/YP_MAP_X_LATE

Related Information

In this book, “Yellow Pages Client Interface” on page 2-914.

The **makedbm**, **ypserv**, **yppush**, and **ypxfr** commands in *AIX Operating System Commands Reference*.

The Yellow Pages section in *Managing the AIX Operating System*

Chapter 4. Miscellaneous Facilities

About This Chapter

This chapter describes miscellaneous facilities, such as macro packages and character set tables.

ascii

Purpose

Maps the ASCII character set.

Synopsis

`cat /usr/pub/ascii`

Description

ASCII is a map of the ASCII character set that gives both the octal and hexadecimal equivalents for each character. This file can be printed as needed.

Note: This is neither the PC ASCII nor the RT ASCII character set. See “data stream” on page 4-5 for information about these character sets. Figure 4-1 contains the octal characters in **ascii**.

000 nul	001 soh	002 stx	003 etx	004 eot	005 enq	006 ack	007 bel
010 bs	011 ht	012 nl	013 vt	014 np	015 cr	016 so	017 si
020 dle	021 dcl	022 dc2	023 dc3	024 dc4	025 nak	026 syn	027 etb
030 can	031 em	032 sub	033 esc	034 fs	035 gs	036 rs	037 us
040 sp	041 !	042 "	043 #	044 \$	045 %	046 &	047 '
050 (051)	052 *	053 +	054 ,	055 -	056 .	057 /
060 0	061 1	062 2	063 3	064 4	065 5	066 6	067 7
070 8	071 9	072 :	073 ;	074 <	075 =	076 >	077 ?
100 @	101 A	102 B	103 C	104 D	105 E	106 F	107 G
110 H	111 I	112 J	113 K	114 L	115 M	116 N	117 O
120 P	121 Q	122 R	123 S	124 T	125 U	126 V	127 W
130 X	131 Y	132 Z	133 [134 \	135]	136 ^	137 _
140 `	141 a	142 b	143 c	144 d	145 e	146 f	147 g
150 h	151 i	152 j	153 k	154 l	155 m	156 n	157 o
160 p	161 q	162 r	163 s	164 t	165 u	166 v	167 w
170 x	171 y	172 z	173 {	174	175 }	176 ~	177 del

Figure 4-1. Octal ASCII Character Set

Figure 4-2 on page 4-4 contains the hexadecimal characters in **ascii**.

ascii

00 nul	01 soh	02 stx	03 etx	04 eot	05 enq	06 ack	07 bel
08 bs	09 ht	0A nl	0B vt	0C np	0D cr	0E so	0F si
10 dle	11 dcl	12 dc2	13 dc3	14 dc4	15 nak	16 syn	17 etb
18 can	19 em	1A sub	1B esc	1C fs	1D gs	1E rs	1F us
20 sp	21 !	22 "	23 #	24 \$	25 %	26 &	27 '
28 (29)	2A *	2B +	2C ,	2D -	2E .	2F /
30 0	31 1	32 2	33 3	34 4	35 5	36 6	37 7
38 8	39 9	3A :	3B ;	3C <	3D =	3E >	3F ?
40 @	41 A	42 B	43 C	44 D	45 E	46 F	47 G
48 H	49 I	4A J	4B K	4C L	4D M	4E N	4F O
50 P	51 Q	52 R	53 S	54 T	55 U	56 V	57 W
58 X	59 Y	5A Z	5B [5C \	5D]	5E ^	5F _
60 `	61 a	62 b	63 c	64 d	65 e	66 f	67 g
68 h	69 i	6A j	6B k	6C l	6D m	6E n	6F o
70 p	71 q	72 r	73 s	74 t	75 u	76 v	77 w
78 x	79 y	7A z	7B {	7C	7D }	7E ~	7F del

Figure 4-2. Hexadecimal ASCII Character Set

File

`/usr/pub/ascii`

data stream

Purpose

Defines the data stream that an **hft** virtual terminal uses in KSR mode.

Description

The IBM RT is capable of addressing 1024 distinct displayable characters. To designate these characters using 8-bit bytes, a code page convention is used. Each **code page** is an ordered set of up to 256 characters, which are called **code points**. The first 32 code points of each code page are reserved for control codes and are the same for all code pages. The control codes do not have graphic representations, so each code page can have a maximum of 224 distinct graphic characters.

The remaining characters are divided into three code pages called P0, P1, and P2. Two additional code pages called USER1 and USER2 are provided for user-defined symbols.

Code points in the range 32 to 127 (0x20 to 0x7F) of code page P0 represent the standard 7-bit US ASCII graphic symbols. P0 code points 128 to 255 (0x80 to 0xFF) and code points in pages P1 and P2 are collectively called **extended characters**.

The following code page maps show the predefined graphic display symbols and their code point values within each of the three code pages.

data stream

		First Hexadecimal Digit															
		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
Second Hexadecimal Digit	0	NUL	DLE	BLANK (SPACE)	0	@	P	'	p	Ç	É	á	⋮	⌞	ø	Ó	-
	1	SOH	DC1	!	1	A	Q	a	q	ü	æ	í	⋮	⌞	Ð	ß	±
	2	STX	DC2	"	2	B	R	b	r	é	Æ	ó	⋮	⌞	Ê	Ô	=
	3	ETX	DC3	#	3	C	S	c	s	â	ô	ú	⋮	⌞	Ë	Ò	¾
	4	EOT	DC4	\$	4	D	T	d	t	ä	ö	ñ	⋮	⌞	È	õ	¶
	5	ENQ	NAK	%	5	E	U	e	u	à	ò	Ñ	Á	⌞	'	Õ	§
	6	ACK	SYN	&	6	F	V	f	v	å	û	ä	Â	ã	Í	µ	÷
	7	BEL	ETB	'	7	G	W	g	w	ç	ù	ó	À	Ã	Î	þ	~
	8	BS	CAN	(8	H	X	h	x	ê	ÿ	ı	©	⌞	Ï	ƒ	°
	9	HT	EM)	9	I	Y	i	y	ë	Ö	®	⌞	⌞	Ú	•	..
	A	LF	SUB	*	:	J	Z	j	z	è	Ü	¬	⌞	⌞	Û	•	
	B	VT	ESC	+	;	K	[k	{	ï	ø	½	⌞	⌞	Ü	¹	
	C	FF	SS4	,	<	L	\	l		î	£	¼	⌞	⌞	Ý	³	
	D	CR	SS3	—	=	M]	m	}	ì	Ø	ı	ç	⌞	ı	Ý	²
	E	SO	SS2	.	>	N	^	n	~	Ä	x	«	¥	⌞	İ	-	■
	F	S1	SS1	/	?	O	_	o	Δ	Å	f	»	⌞	⌞	'	BLANK 'FF'	

Figure 4-3. Code Page P0

		First Hexadecimal Digit															
		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
Second Hexadecimal Digit	0	NUL DLE	•	►	ã	ý	Ò	Ě	š	ť	ł	Ē	Į	ō	u	“	
	1	SOH DC1	☺	◄	ß	þ	Ó	Č	Ĺ	ř	'n	g	ij	Ō	U	”	
	2	STX DC2	☺	↕	Â	-	Õ	Ć	Ń	ő	S	ĝ	IJ	œ	ŵ	=	
	3	ETX DC3	♥	!!	À	⌘	³	~	Ř	ů	-	Ĝ	Ĵ	Œ	Ŵ	-	
	4	EOT DC4	♦	¶	Á	Đ	Ů	Ę	Š	Ť	Ł	Ġ	Ĵ	ŕ	ŷ	+	
	5	ENQ NAK	♣	§	Ã	Ý	Ù	Ů	·	Ř	Ť	Ġ	ķ	Ŕ	Ÿ	∞	
	6	ACK SYN	♠	■	ø	Ɔ	Ú	Ď	ž	Ŏ	ā	G	K	š	ÿ	π	
	7	BEL ETB	•	↕	Ê	®	ą	Ł	Ł	Ů	Ā	ĥ	ƚ	Ŝ	©	Δ	
	8	BS CAN	•	↑	Ë	³ / ₄	ě	Ĭ	Ž	ǎ	ĉ	Ĥ	ł	ţ	'	→	
	9	HT EM	○	↓	È	-	č	ň	ž	ğ	Ĉ	h	Ł	Ŧ	™	/	
	A	LF SUB	○	→	Í	·	ć	đ	ž	İ	\	H	I	ũ	¹ / ₈	†	
	B	VT ESC	♂	←	Î	'	ę	ř	Ž	Ǻ	ċ	ĩ	L	Ũ	³ / ₈	<	
	C	FF SS4	♀	└	Ï	=	ű	ś	Ž	Ğ	Ċ	ĩ	ņ	ũ	⁵ / ₈	>	
	D	CR SS3	♪	↔	Ì	õ	ď	°	Ł	˘	è	ī	N	Ū	⁷ / ₈	℞	
	E	SO SS2	♪	▲	Ø	'	í	ł	Ń	"	È	Ī	ŋ	ū	×	ē	
	F	S1 SS1	☀	▼	ø	Ô	A	ń	Š	ş	ē	ł	Ŋ	Ū	'	∴	

Figure 4-4. Code Page P1

data stream

		First Hexadecimal Digit															
		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
Second Hexadecimal Digit	0	NUL	DLE	↗	⊂	ω	1	∂	6	⏏	τ	√					
	1	SOH	DC1	↘	⊕	v	2	~	7	⏏	⌚	⏏					
	2	STX	DC2	‡	⌒	o	3	□	8	⏏	θ	n					
	3	ETX	DC3	≠	⊗	ρ	4	■	9	⏏	Ω						
	4	EOT	DC4	√	"	γ	5	⏏	φ	⏏	δ	¢					
	5	ENQ	NAK	^	=	θ	6	⏏	ρ↑s	⏏	∞	⏏					
	6	ACK	SYN		Ψ	⏏	7	⏏	⏏	⏏	∅	√					
	7	BEL	ETB	⌒	ε	⏏	8	⏏	⏏	⏏	ε	↓					
	8	BS	CAN	<	λ	≅	9	α	⏏	■	n	∅					
	9	HT	EM	>	η	ξ	⏏	Δ	⏏	■	≡						
	A	LF	SUB	+	ι	χ	⏏	Y	⏏	α	≥						
	B	VT	ESC	□	⏏	v	Ψ	≈	⏏	β	≤						
	C	FF	SS4	'	⏏	ζ	Π	~	⏏	Γ	∫						
	D	CR	SS3	∫	%o	∫	Λ	°	⏏	π	∫						
	E	SO	SS2	U	ℳ	λ	●	4	⏏	Σ	≈						
	F	S1	SS1	⏏	κ	o	b	5	⏏	σ	●						

Figure 4-5. Code Page P2

Code Page Switching

Characters from code page P0 are represented in a character data stream by a single 8-bit byte corresponding to their code points.

Characters from other code pages are selected with single-shift controls. A *single-shift control* is one of the single-byte control codes SS1 (0x1F), SS2 (0x1E), SS3 (0x1D), and SS4 (0x1C). Each of these codes indicates that the following byte specifies a character from a code page other than P0. These control codes are called “single shifts” because they shift to another code page for a single character; that is, they are nonlocking shifts.

The byte that follows a single shift corresponds to the code point for the desired character, but with the most significant bit set. In other words, SS1, SS2, SS3, and SS4 must be followed by a byte in the range 0x80 to 0xFF. A single shift followed by 0x00 to 0x7F is not a valid code sequence. The single shift that is used specifies the upper or lower half of a code page as follows:

SS1	Lower half of code page P1 (P1 0x20 to 0x7F)
SS2	Upper half of code page P1 (P1 0x80 to 0xFF)
SS3	Lower half of code page P2 (P2 0x20 to 0x7F)
SS2	Upper half of code page P2 (P2 0x80 to 0xFF).

Note that in this scheme, code points in the range 0x00 to 0x7F (7-bit US ASCII) are unique in the data stream, and that they are never validly preceded by a single-shift control. This encoding scheme minimizes the changes necessary to existing software that is oriented toward 7-bit ASCII.

If a single-shift control is followed by a byte with the most significant bit set to zero (that is, a byte in the range 0x00 to 0x7F), then the single-shift prefix is ignored, and the byte is processed as an unprefixed character.

On both input and output, graphic character code points that are *not* prefixed with a single-shift control select a display symbol from the active graphic display set (G0 or G1) to be echoed or displayed on the screen. By default, both G0 and G1 are set to P0, and G0 is the active display set. The active graphic display set can be set to G0 or G1 with the SI and SO single-byte controls, respectively (see “Single-Byte Controls” on page 4-11). The mapping used for G0 and G1 can be set with the SG0 and SG1 control sequences (see “Multibyte Controls” on page 4-13).

On both input and output, valid graphic character code points that *are* prefixed with SS1, SS2, SS3, or SS4 bypass the active graphic display set and echo or display characters directly from code page P1 or P2.

Nonspacing Characters

For convenience when typing diacritical (accented) characters, a nonspacing or “dead” character facility is provided. A **nonspacing character sequence** is a two-key sequence consisting of one of the 13 diacritics followed by an alphabetic character or a space. The virtual terminal subsystem converts this two-key sequence into a single code point that may have a single-shift prefix. The resulting character is the alphabetic character with the specified diacritic mark. A diacritic followed by a space translates to the diacritic character itself.

The 13 valid diacritics are:

'	Acute Accent or Apostrophe	0xEF or 0x27
`	Grave Accent	0x60
^	Circumflex Accent	0x5E
..	Umlaut Accent	0xF9
~	Tilde Accent	0x7E
ˇ	Caron Accent	0x1FF3
˘	Breve Accent	0x1E9D
¨	Double Acute Accent	0x1E9E
°	Overcircle Accent	0x1FFD
·	Overdot Accent	0x1E85
—	Macron Accent	0x1EA3
¸	Cedilla Accent	0xF7
ł	Ogonek Accent	0x1E87

If a nonspacing character and the following character do not combine to form a diacritical character in the set of predefined graphic symbols, then the diacritic is treated as a separate character code. For example, ~Q is treated as two characters, ~ and Q.

Note that nonspacing characters apply only to keyboard input and are not a feature of the data stream used by applications. Also, a diacritic must be explicitly designated as being nonspacing in the keyboard mapping for this facility to operate. None of the keys on the standard U.S. keyboard mapping are defined as nonspacing characters. However, nonspacing characters can be defined. See “Set Keyboard Map (HFSKBD)” on page 5-53 for details.

Controls

Two types of controls are valid in a character stream data:

- Single-byte controls (also called **control characters** and **control codes**), which have character values from 0 to 31 (0x00 to 0x1F)
- Multibyte controls, which are also called escape sequences and control sequences.

Single-Byte Controls

The single-byte controls are common to all code pages. The following list shows the single-byte controls and their interpretation in KSR coded data. A line introducing each control gives its mnemonic, its code value, and its function.

- NUL, 0x00, (Null) has no terminal function.
- SOH, 0x01, (Start of Header) has no terminal function.
- STX, 0x02, (Start of Text) has no terminal function.
- ETX, 0x03, (End of Text) has no terminal function.
- EOT, 0x04, (End of Transmission) has no terminal function.
- ENQ, 0x05, (Enquiry) has no terminal function.
- ACK, 0x06, (Acknowledge) has no terminal function.
- BEL, 0x07, (Bell) causes an audible alarm to sound.
- BS, 0x08, (Backspace) moves the cursor position to the left one column, unless the cursor is at the left boundary of the presentation space. In that case, the cursor position does not change.
- HT, 0x09, (Horizontal Tab) moves the cursor position forward to the next tab stop. If the cursor is already in the last column of a line, then the cursor position does not change. Note that the CHT (cursor horizontal tab) multibyte control performs a similar operation, but also performs line wrapping.
- LF, 0x0A, (Line Feed) if the LNM mode is reset, the line feed moves the cursor position down one line. If the LNM mode is set (default), the line feed is treated as a NEL and moves the cursor position to the first position of the next line. In either case, if the cursor is already on the last line of the PS, the PS lines scroll up one line. The top line of the PS disappears and a blank line is inserted as the new bottom line.
- VT, 0x0B, (Vertical Tab) moves the cursor position down to the next line that is defined as a vertical tab stop. Tabs stops are always set at the first and last lines of the PS. If the cursor was already on the last line of the PS and HFWRAP mode is not set, the cursor stays on the last line in the PS. If HFWRAP mode is set, the cursor moves to the top line in the PS. The column position does not change in any case.
- FF, 0x0C, (Form Feed) treated as a line end; see NEL.
- CR, 0x0D, (Carriage Return) if the CNM mode is reset (default), the carriage return moves the cursor position to the first character of the line indicated by the cursor. If the CNM mode is set, the carriage return is treated as an NEL and causes the cursor position to move to the first position of the next line. In this case, if the cursor is already on the last line of the PS, the PS lines scroll up one line. The top line of the PS disappears and a blank line is inserted as the new bottom line.

data stream

- SO, 0x0E, (Shift Out) maps the subsequently received graphic codes to display symbols according to the active G1 character set. See “display symbols” on page 4-26 for a list of the display symbols.
- SI, 0x0F, (Shift In) maps the subsequently received graphic codes to display symbols according to the active G0 character set. See “display symbols” on page 4-26 for a list of the display symbols.
- DLE, 0x10, (Data Link Escape) has no terminal function.
- DC1, 0x11, (Device Control 1) has no terminal function when output.
- DC2, 0x12, (Device Control 2) has no terminal function.
- DC3, 0x13, (Device Control 3) has no terminal function when output.
- DC4, 0x14, (Device Control 4) has no terminal function.
- NAK, 0x15, (Negative Acknowledgment) has no terminal function.
- SYN, 0x16, (Synchronous) has no terminal function.
- ETB, 0x17, (End of Block) has no terminal function.
- CAN, 0x18, (Cancel) has no terminal function.
- EM, 0x19, (End of Medium) has no terminal function.
- SUB, 0x1A, (Substitute) has no terminal function.
- ESC, 0x1B, (Escape) defines the beginning of a multibyte control sequence as defined in “Multibyte Controls” on page 4-13.
- SS4, 0x1C, (Single Shift 4) causes the following byte to be interpreted as belonging to the upper half of code page P2 (see “Code Page Switching” on page 4-9).
- SS3, 0x1D, (Single Shift 3) causes the following byte to be interpreted as belonging to the lower half of code page P2.
- SS2, 0x1E, (Single Shift 2) causes the following byte to be interpreted as belonging to the upper half of code page P1.
- SS1, 0x1F, (Single Shift 1) causes the following byte to be interpreted as belonging to the lower half of code page P1.
- DEL, 0x7F, (Delete) has no terminal function.

Multibyte Controls

This section defines the code points and effects on the virtual terminal for multibyte control sequences that are recognized in KSR mode. All of them begin with the ESC code (0x1B) followed by a [(0x5B) and include all subsequent bytes up to and including the first code in the range 0x40–0x7F. Any multibyte control sequences not defined below are ignored. Invalid sequences return an error Device Status Report to the program. Multibyte control sequences of more than 16 codes are considered invalid on receipt of the 17th code. The next code is not considered a part of that sequence. Also, numeric parameters in control sequences contain no more than 3 digits. The numeric value of the parameter may be incorrect if more than three digits are used, and the numeric value never exceeds 255.

Controls effect a virtual terminal's presentation space (PS) and its related cursor (pointer into the PS). The presentation space is a logical array of display symbols, N columns by M lines.

The following list gives the valid multibyte control code sequences. A line introducing each control gives its mnemonic, its code sequence, and its function. The code sequence is shown in terms of ASCII characters. For example, the sequence ESC A represents two codes with a value of 0x1B41.

- CBT ESC [*PN* Z Cursor Back Tab

Moves the cursor back the number of horizontal tab stops specified by *PN*. Tab stops are always set at the first and last columns of each line. If the cursor is already in the first column of a line and HFWRAP mode is set, the cursor moves to the last column. If AUTONL is also set, the cursor moves to the last column of the previous line. In this case, if the cursor is already on the first row of the PS, it moves to the last row.

- CHA ESC [*PN* G Cursor Horizontal Absolute

Moves the cursor to the column specified by *PN*, unless the column exceeds the PS width. If the column exceeds the PS width, the cursor moves to the PS column farthest to the right.

- CHT ESC [*PN* I Cursor Horizontal Tab

Moves the cursor position forward to the *PN*th following tab stop. If the cursor is already in the last column of a line and HFWRAP mode is set, then the cursor returns to the first column of the line. If AUTONL mode is also set, then the cursor moves to the first column of the next line. In this case, if the cursor is already on the last line of the PS, then the cursor moves to the first column of the first line. Note that the HT (horizontal tab) single-byte control does not cause wrapping to occur.

- CTC ESC [*PS* W Cursor Tab Stop Control

- 0 Set a horizontal tab at cursor
- 1 Set a vertical tab at cursor
- 2 Clear a horizontal tab at cursor
- 3 Clear a vertical tab at cursor

- 4 Clear all horizontal tabs on line
- 5 Clear all horizontal tabs
- 6 Clear all vertical tabs

Sets or clears one or more tabulation stops according to the parameter specified. Tab stops on the first or last column cannot be cleared. When horizontal tab stops are set or cleared, the number of lines affected is all (if Tabulation Stop Mode is set) or one (if Tabulation Stop Mode is reset). This control does not change the position of characters already in the presentation space.

- CNL ESC [*PN* E Cursor Next Line

Moves the cursor down the number of lines specified by *PN*, and over to the first position of that line. If the cursor was already on the bottom PS line and HFWRAP mode is not set, it is positioned at the beginning of that line. If HFWRAP mode is set, the cursor wraps from the bottom line to the top PS line.

- CPL ESC [*PN* F Cursor Preceding Line

Moves the cursor back the number of lines specified by *PN*, and over to the first position of that line. If the cursor was already on the top PS line and HFWRAP mode is not set, the cursor is positioned at the beginning of that line. If HFWRAP mode is set, the cursor wraps from the top line to the bottom line of the PS.

- CPR ESC [*PN* ; *PN* R Cursor Position Report

Reports the current cursor position. The first numeric parameter is the line number, and the second is the column. Line and column values are sent to the application as information. However, if the information is received by the virtual terminal, it is treated as a CUP control.

- CUB ESC [*PN* D Cursor Backward

Moves the cursor backward on the line the specified number of columns. If this cursor movement exceeds the left PS boundary and HFWRAP mode is not set, the cursor stops at the leftmost PS position. If HFWRAP mode is set, the cursor wraps from the leftmost column to the rightmost column of the preceding PS line. In HFWRAP mode the cursor also wraps from the home to the rightmost bottom position of the PS.

- CUD ESC [*PN* B Cursor Down

Moves the cursor down the number of lines specified by *PN*. If this cursor movement exceeds the bottom PS boundary and HFWRAP mode is not set, the cursor stops on the last PS line. If HFWRAP mode is set, the cursor wraps from the bottom line to the top line of the PS.

- CUF ESC [*PN* C Cursor Forward

Moves the cursor forward on the line the specified number of columns. If this cursor movement exceeds the right PS boundary and HFWRAP mode is not set, the cursor stops at the rightmost PS position. If HFWRAP mode is set, the cursor wraps from the rightmost column to the leftmost column of the following line in the PS. In HFWRAP

mode, the cursor also wraps from rightmost bottom position to the home position of the PS.

- CUP ESC [*PN* ; *PN* H Cursor Position

Moves the cursor to the line specified by the first parameter, and to the column specified by the second parameter. If this movement crosses a PS boundary, the cursor stops at the PS boundary.

- CUU ESC [*PN* A Cursor Up

Moves the cursor up the specified number of lines. If this cursor movement exceeds the top PS boundary and HFWRAP mode is not set, the cursor stops on the first PS line. If HFWRAP mode is set, the cursor wraps from the top line to the bottom line in the PS.

- CVT ESC [*PN* Y Cursor Vertical Tab

Moves the cursor down the number of vertical tab stops specified. Tab stops are assumed at the top and bottom PS lines. If there are not enough vertical tab stops in the PS and HFWRAP mode is not set, the cursor stops on the last line in the PS. If HFWRAP mode is set, the cursor wraps from the bottom line to the top line of the PS.

- DCH ESC [*PN* P Delete Character

Deletes the cursor character and the following *PN*-1 characters on the line indicated by the cursor. The characters following the deleted characters on the line overlay the deleted character positions. The line is cleared from the end of the line to the edge of the presentation space. If the number of characters to be deleted exceeds the number of columns from the cursor to the PS right boundary, then all the characters from the cursor to the PS boundary are replaced with empty spaces and a DSR control sequence identifying an error is returned to the application.

- DL ESC [*PN* M Delete Line

Deletes the line and the *PN*-1 following lines in the PS. The lines following the deleted lines are scrolled up *PN* lines and *PN* blanks lines are placed at the bottom of the PS. If there are less than *PN* lines from the line indicated by the cursor to the bottom of the PS, the line indicated by the cursor and all the following PS lines are replaced with empty lines.

- DSR ESC [*PN* n Device Status Report Request

6 Request Cursor Position Report

13 Error Report

A request cursor position report (CPR) sends a cursor position report from the virtual terminal to the application. An error report is sent from the virtual terminal to the application when the virtual terminal receives an invalid control sequence. Error reports are private reports which conform to the ANSI standard for private parameters.

data stream

- DMI ESC ' (left quote) Disable Manual Input

This control, when received in an output data stream, causes keyboard input to this terminal to be ignored. This control is ignored when received from the keyboard.

- EMI ESC b Enable Manual Input

This control, when received in an output data stream, restarts keyboard input recognition and buffering if previously disabled with a DMI multibyte control. This control is ignored when received from the keyboard.

- EA ESC [0 0 Erase to End of Area
 ESC [1 0 Erase from Start of Area
 ESC [2 0 Erase All of Area.

This control is treated like an EL control sequence.

- ED ESC [0 J Erase to End of Display
 ESC [1 J Erase from Start of Display
 ESC [2 J Erase All of Display.

Erases certain characters within the PS. Erased characters are replaced with empty spaces. Erase to end of display erases the character indicated by the cursor and all following characters in the PS. Erase from start of display erases the first character of first line and the following characters up to and including the character indicated by the cursor. Erase all of display erases all the characters on the PS.

- EF ESC [0 N Erase to End of Field
 ESC [1 N Erase from Start of Field
 ESC [2 N Erase All of Field.

Erases certain characters between horizontal tab stops. Erased characters are replaced with empty spaces. Erase to end of field erases the character indicated by the cursor and all following characters before the next tab stop. Erase from start of field erases the character at the tab stop preceding the cursor and the following characters up to and including the character indicated by the cursor. Erase all of field erases the character at the tab stop preceding the cursor, and the following characters up to and including the character at the tab stop following the cursor. Tab stops are assumed at the first and last columns of the PS when executing this control.

- EL ESC [0 K Erase to End of Line
 ESC [1 K Erase from Start of Line
 ESC [2 K Erase All of Line.

Erases certain characters within a line. Erased characters are replaced with empty spaces. Erase to end of line erases the character indicated by the cursor and all

following characters on the line. Erase from start of line erases the first character of first line and the following characters up to and including the character indicated by the cursor. Erase all of line erases all the characters on the line.

- ECH ESC [*PN* X Erase Character

Erases the character indicated by the cursor and the following *PN*-1 characters on that line. Erased characters are replaced with empty spaces. If there are less than *PN* characters from the cursor to the PS right boundary, then the character indicated by the cursor and all the following characters on the line are replaced empty spaces.

- HTS ESC H Horizontal Tab Stop

Sets a horizontal tab stop at the current horizontal position. If TSM is set, then the tab stop applies only to this line. If TSM is reset, then the tab stop applies to all PS lines. This control does not change the positioning of characters already in the presentation space.

- HVP ESC [*PN* ; *PN* f Horizontal and Vertical Position

Moves the cursor to the line specified by the first parameter, and to the column specified by the second parameter. If this movement would cross a PS boundary, the cursor stops at the current PS boundary.

- ICH ESC [*PN* @ Insert Character

Inserts *PN* empty spaces before the character indicated by the cursor. The string of characters starting with the character indicated by the cursor and ending with last character of the line are shifted *PN* columns to the right. Characters shifted past the PS right boundary are lost. The cursor does not move.

- IL ESC [*PN* L Insert Line

Inserts *PN* empty lines before the line indicated by the cursor. The line indicated by the cursor is scrolled down. The cursor position on the screen is not affected.

- IND ESC D Index

Moves cursor down one line. If the cursor was already on the bottom line of the PS, then the top line is lost, the other lines move up one line, and a blank line becomes the new bottom line.

- NEL ESC E Next Line

Moves the cursor to the first position of the following line. If the cursor was already on the bottom line of the PS, then the top line is lost, the other lines move up one, and a blank line becomes the new bottom line.

- KSI ESC [*PS* p Keyboard Status Information

The virtual terminal generates this control whenever **HFHOSTS** and **HFXLATKBD** are set and the status of the keyboard changes. Each selective parameter is the character-coded decimal value of a keyboard status byte. For example, if the keyboard

has two status bytes, the control sequence is **ESC [xxx;yyy p**, where *xxx* is the value of the high-order byte and *yyy* is the value of the low-order byte. This is a private control that conforms to the ANSI standards for private control sequences. The virtual terminal display handler ignores this sequence whether it is received from the application or echoed. The values of the status bytes are described in “Untranslated Key Control” on page 5-73.

- **PFK** **ESC [PN q** PF Key Report

The control sequence is sent by the virtual terminal to the application when a program function key (PFK) code is received from the keyboard. The parameter *PN* is a PF key number from 1 to 255. This is a private control that conforms to the ANSI standards for private control sequences. This sequence is ignored by the virtual terminal display handler whether received from the application or echoed.

- **RCP** **ESC [u** Restore Cursor Position

Moves the cursor to the position saved by the last SCP control. If no SCP has been received, then the cursor position is set to the first character of the first line. This is a private control that conforms to the ANSI standards for private controls. This control has no terminal function when received from the keyboard.

- **RI** **ESC L** Reverse Index

Moves the cursor up one line, unless the cursor is already on the PS top line. In that case, if HFWRAP mode is not set, then the cursor does not move. If HFWRAP mode is set, the cursor moves to the bottom line of the PS. The column position does not change.

- **RIS** **ESC c** Reset to Initial State

Resets the virtual terminal to the state of a newly-opened virtual terminal: erases all PS data, places the cursor at the home position, resets graphic rendition to normal, resets subscripting and superscripting, and sets tab stops, modes, keyboard map, character maps and echo maps to their default values.

Note: The RIS multibyte control resets the VRM virtual terminal defaults, which are not necessarily the same as the defaults of an HFT device.

- **RM** **ESC [PS l** Reset Mode

20 LNM - Line Feed - New Line Mode

4 IRM - Insert Mode

12 SRM - Send Receive Mode (set ECHO off)

18 TSM - Tabulation Stop Mode

?21 CNM - Carriage Return - New Line Mode

?7 AUTONL - Wrap character to following line when end of current line reached

Resets the modes specified in the parameter string. Multiple parameters must be separated by semicolons. The modes that can be reset are listed above with the appropriate parameter code. All other mode parameters are ignored.

TSM mode determines whether horizontal tabs apply identically to all line (TSM reset) or uniquely to each line on which they are set (TSM set).

- SCP ESC [s Save Cursor Position

Saves the current cursor position. Any previously saved cursor position is lost. The cursor can be restored to this position with an RCP control. This is a private control that conforms to the ANSI standards for private controls. This control has no terminal function when received from the keyboard.

- SD ESC [*PN* T Scroll Down

Moves all the PS lines down *PN* lines. The bottom *PN* lines are lost, and *PN* empty lines are put at the top of the presentation space. Physical cursor position does not change due to the scroll.

- SL ESC [*PN* SP @ Scroll Left

Moves all the PS characters *PN* column positions to the left. The characters in the *PN* leftmost PS columns are lost, and empty spaces are put in the rightmost *PN* columns of all lines. Physical cursor position does not change due to the scroll.

- SR ESC [*PN* SP A Scroll Right

Moves all the PS characters *PN* column positions to the right. The characters in the *PN* rightmost PS columns are lost, and empty spaces are put in the leftmost *PN* columns of all lines. Physical cursor position does not change due to the scroll.

- SU ESC [*PN* S Scroll Up

Moves all the PS lines up *PN* lines. The top *PN* lines are lost, and *PN* empty lines are put at the bottom of the presentation space. The physical cursor position does not change due to the scroll.

- SGR ESC [*PS* m Set Graphic Rendition

- 0 Normal (none of attributes 1-9)
- 1 Bold or Bright
- 4 Underscore
- 5 Slow Blink
- 7 Negative (reverse image)
- 8 Canceled On (invisible: set to background color)
- 10 Primary Font
- 11 First Alternate Font
- 12 Second Alternate Font
- 13 Third Alternate Font
- 14 Fourth Alternate Font
- 15 Fifth Alternate Font
- 16 Sixth Alternate Font
- 17 Seventh Alternate Font
- 30 Color palette entry 0 foreground

data stream

31 Color palette entry 1 foreground
32 Color palette entry 2 foreground
33 Color palette entry 3 foreground
34 Color palette entry 4 foreground
35 Color palette entry 5 foreground
36 Color palette entry 6 foreground
37 Color palette entry 7 foreground
40 Color palette entry 0 background
41 Color palette entry 1 background
42 Color palette entry 2 background
43 Color palette entry 3 background
44 Color palette entry 4 background
45 Color palette entry 5 background
46 Color palette entry 6 background
47 Color palette entry 7 background
90 Color palette entry 8 foreground
91 Color palette entry 9 foreground
92 Color palette entry 10 foreground
93 Color palette entry 11 foreground
94 Color palette entry 12 foreground
95 Color palette entry 13 foreground
96 Color palette entry 14 foreground
97 Color palette entry 15 foreground
100 Color palette entry 8 background
101 Color palette entry 9 background
102 Color palette entry 10 background
103 Color palette entry 11 background
104 Color palette entry 12 background
105 Color palette entry 13 background
106 Color palette entry 14 background
107 Color palette entry 15 background.

Causes the next characters received in the data stream or from the keyboard to have the display attributes specified by the parameter string. Any parameter not listed above is ignored.

The attributes corresponding to parameters 1 through 9 are cumulative. For example, specifying **underscore** and then specifying **blink** causes following characters to be underscored and blink. To reset one of these attributes, specify **normal** and then reinstate the desired parameters. Multiple parameters are processed in the order listed.

Whether the characters really have the requested attributes on the display depends on the capabilities of the physical display device used by the virtual terminal.

Note that switching between loaded fonts with the SGR sequence causes no data loss, but loading new fonts does cause data loss. (See "Untranslated Key Control" on page 5-73 for more information.)

Characters that cannot be displayed do not exist in the system.

- SG0A ESC (f Set G0 Character Set
 - SG0B ESC , f Set G0 Character Set (Alternate form)
 - : Unique One (User-defined)
 - ; Unique Two (User-defined)
 - < P0 (Display Symbols 32-255)
 - = P1 (Display Symbols 256-479)
 - > P2 (Display Symbols 480-703)
 - ? User1 (Display Symbols 704-927)
 - @ User2 (Display Symbols 928-1023)

Designates the set of characters to use as the G0 set when the G0 set is invoked by SI. The default G0 set is the 224-character code page P0. Unique One and Unique Two may have unique definitions for each virtual terminal. When a virtual terminal is opened, these two sets are equivalent to the < (less than) character. See "Character Set Definition" on page 5-86 about defining Unique One and Unique Two.

- SG1A ESC) f Set G1 Character Set
 - SG1B ESC - f Set G1 Character Set (Alternate)
 - : Unique One (User-defined)
 - ; Unique Two (User-defined)
 - < P0 (Display Symbols 32-255)
 - = P1 (Display Symbols 256-479)
 - > P2 (Display Symbols 480-703)
 - ? User1 (Display Symbols 704-927)
 - @ User2 (Display Symbols 928-1023)

Designates the set of characters to use as the G1 set when the G1 set is invoked by SO. The default G1 set is the 224-character code page P0. Unique One and Unique Two may have unique definitions for each virtual terminal. When a virtual terminal is opened, these two sets are equivalent to <. See "Character Set Definition" on page 5-86 about defining Unique One and Unique Two.

- SM ESC [PS h Set Mode
 - 2 0 LNM - Line Feed - New Line Mode (default = 1)
 - 4 IRM - Insert Replace Mode (default = 0)
 - 1 2 SRM - Send Receive Mode (set echo off) (default = 0)
 - 1 8 TSM - Tabulation Stop Mode (default = 0)
 - ? 2 1 CNM - Carriage Return - New Line Mode (default = 0)
 - ? 7 AUTONL - Wrap to next line when end of line reached (default = 1)

Sets the modes specified in the parameter string. Multiple parameters must be separated by semicolons. The modes that can be set are listed above with the appropriate parameter code. All other mode parameters are ignored.

SRM mode affects translated keyboard input handling. If SRM mode is set, translated keyboard input is never echoed by the virtual terminal, but is immediately returned to the application.

TSM mode determines whether horizontal tabs apply to all lines identically (TSM reset) or if horizontal tabs apply uniquely to each line on which they are set (TSM set).

- TBC ESC [*PS* g Tabulation Clear
 - 0 Horizontal tab at cursor column
 - 1 Vertical tab at line indicated by the cursor
 - 2 Horizontal tabs on line
 - 3 Horizontal tabs in presentation space
 - 4 Vertical tabs in presentation space.

Clears tabulation stops specified by the parameters. Horizontal tab changes affect only the line indicated by the cursor if TSM is set, and horizontal tab changes affect all lines if TSM is reset. Any parameters not listed above are ignored. This control does not change the positioning of characters already in the presentation space.

- VTA ESC [r Virtual Terminal Addressability

This private control sequence precedes a binary header and associated data that provide status information on the IBM 5081 Display Adapter.

- VTD ESC [x Virtual Terminal Data

This private control sequence precedes a binary header and associated data. The block of data can be in formats other than character-coded data, such as binary format. See "General Output" on page 5-78 for details about how this control sequence is used.

- VTL ESC [y Virtual Terminal Device Input

This private control sequence precedes binary format input data from a mouse, tablet, LPFK, or valuator device. See "Input Device Report" on page 5-74 for details about how this control sequence is used.

- VTR ESC [w Virtual Terminal Raw Keyboard Input

This private control sequence precedes raw (untranslated) keyboard input data, which is in a binary format. See "Untranslated Key Control" on page 5-73 for details about how this control sequence is used.

- VTS ESC I Vertical Tab Stop

Sets a vertical tab stop at the line indicated by the cursor. This control does not change the positioning of characters already in the presentation space.

File

`/usr/pub/ibmcharset` Contains RT character set.

Related Information

In this book: “display symbols” on page 4-26 and “hft” on page 5-39.

Keyboard Description and Character Reference.

“Overview of International Character Support” in *Managing the AIX Operating System*.

dirent.h

dirent.h

Purpose

Describes the format of a directory entry, without regard to file-system type.

Synopsis

```
#include <sys/dirent.h>
#include <sys/param.h>
#include <sys/type.h>
```

Description

The **dirent** data structure is defined in the **dirent** header file and describes the format of a directory entry, without reference to the type of underlying file system.

The **dirent** structure is used in the directory access operations documented in “directory” on page 2-149. Using these access operations, **dirent**, and the structures, constants, and macros described below does the following:

- Shields you from the details of implementing a directory
- Provides a consistent interface to directories across all types of file systems.

Note: The type **DIR**, defined with a **typedef**, is a structure that contains information about a directory. In general, only the directory access operations use the information in the **DIR** structure.

The **dirent** structure is defined in the **dirent.h** header file, and it contains the following members:

```
    ulong  d_ino;                /* inode number of entry */
    ushort d_reclen;             /* length of this entry */
    ushort d_namlen;             /* length of string in d_name
    char    d_name[NAME_MAX+1]; /*name of entry (filename) */
```

The constant **NAME_MAX** is the maximum number of bytes in a file name, not including the terminating null byte. Related to this constant is **PATH_MAX**, which is the maximum number of bytes in the full path name of a file, also not including the terminating null byte.

Note the following:

- These are maximum file names and path names defined across all types of file systems.
- Each file system can define constants applicable only to that specific file system. For example, the AIX file system defines **DIRSIZ** as the maximum number of bytes in a file name.

Note: Using file-system specific constants and directory structures makes it very difficult to port code across different types of file systems.

The size of a **dirent** structure varies, depending on the number of bytes in the file name. The minimum size of a **dirent** structure is computed using the following macro:

```
DIRENTSIZ (dp)  
dirent *dp
```

Related Information

In this book: “directory” on page 2-149.

The discussion of Japanese Language Support in *Japanese Language Support User's Guide*.

display symbols

display symbols

Purpose

Defines the set of character symbols that can be displayed on an HFT display device in KSR mode.

Description

Each character code passed in KSR data is translated into one of 1024 10-bit display symbol codes. Codes 0 through 703 (0x2bf) are predefined to be common across all virtual terminals. Codes 704 (0x2c0) through 1023 (0x3ff) are reserved for user-defined extensions to the display symbol set. Display symbols 0 through 31 (0x1f) represent control functions and have no graphic representations.

Code pages P0, P1, and P2 contain all of the predefined characters (see “code pages” 4-5). The first 32 code points of each are reserved for control characters and are common to all three code pages. The remaining characters are divided between P0, P1, and P2. Thus, each code page can have up to 224 distinct graphic characters.

In addition to the predefined code pages P0, P1, and P2, you can define two code pages called *Unique One* and *Unique Two*. See “fonts” on page 3-79, “data stream” on page 4-5, and “Reconfigure (HFRCONF)” on page 5-48 for information you need to define such character sets.

The columns of the following tables represent:

Font Position

The position of the graphic display symbol within the font definition.

Code Page/Code Point

The code page of the symbol and the offset within that code page.

char String

The internal hexadecimal representation as a string of type **char**, including the single-shift control for characters in code pages other than P0.

NLchar Value

The value of the **NLchar** data type that corresponds to the character. The values 256–287 and 512–543 are not listed in this table because they correspond to control codes in code pages P1 and P2. See “NLchar” on page 2-514 for more information about this data type.

NCesc Esc Seq

The ASCII character or escape sequence that corresponds to the character after being translated by the **NCesc** macro. The **NLchar** values 256–287 and 512–543, which

correspond to control codes in code pages P1 and P2, translate to \< >, where two space characters (0x20) appear between the angle brackets. **NLchar** values outside the valid range translate to \<?>. See "conv" on page 2-105 and "NLescstr, NLunesctr, NLflatstr" on page 2-516 for related information.

The first table begins at font position 32 because the first 32 positions are reserved for the single-byte controls. The IBM PC ASCII graphic symbols for positions 1 through 31 are located at positions 257 through 287 and are not in any way associated with single-byte control functions.

Font Position	Character	Code Page Code Point	char String	NLchar Value	NCesc Esc Seq
32	Space	P0 32 (0x20)	0x20	32	Space
33	! Exclamation Point	P0 33 (0x21)	0x21	33	!
34	" Double Quotation Mark	P0 34 (0x22)	0x22	34	"
35	# Pound Sign	P0 35 (0x23)	0x23	35	#
36	\$ Dollar Sign	P0 36 (0x24)	0x24	36	\$
37	% Percent Sign	P0 37 (0x25)	0x25	37	%
38	& Ampersand	P0 38 (0x26)	0x26	38	&
39	' Apostrophe, Acute Accent	P0 39 (0x27)	0x27	39	'
40	(Left Parenthesis	P0 40 (0x28)	0x28	40	(
41) Right Parenthesis	P0 41 (0x29)	0x29	41)
42	* Asterisk	P0 42 (0x2a)	0x2a	42	*
43	+ Plus Sign	P0 43 (0x2b)	0x2b	43	+
44	, Comma	P0 44 (0x2c)	0x2c	44	,
45	- Hyphen, Minus Sign	P0 45 (0x2d)	0x2d	45	-
46	. Period	P0 46 (0x2e)	0x2e	46	.
47	/ Slash	P0 47 (0x2f)	0x2f	47	/
48	0 Zero	P0 48 (0x30)	0x30	48	0
49	1 One	P0 49 (0x31)	0x31	49	1
50	2 Two	P0 50 (0x32)	0x32	50	2
51	3 Three	P0 51 (0x33)	0x33	51	3
52	4 Four	P0 52 (0x34)	0x34	52	4
53	5 Five	P0 53 (0x35)	0x35	53	5
54	6 Six	P0 54 (0x36)	0x36	54	6
55	7 Seven	P0 55 (0x37)	0x37	55	7
56	8 Eight	P0 56 (0x38)	0x38	56	8

Figure 4-6 (Part 1 of 8). Code Page P0

display symbols

Font Position	Character	Code Page Code Point	char String	NLchar Value	NCesc Esc Seq
57	9 Nine	P0 57 (0x39)	0x39	57	9
58	: Colon	P0 58 (0x3a)	0x3a	58	:
59	; Semicolon	P0 59 (0x3b)	0x3b	59	;
60	< Less Than Sign	P0 60 (0x3c)	0x3c	60	<
61	= Equal Sign	P0 61 (0x3d)	0x3d	61	=
62	> Greater Than Sign	P0 62 (0x3e)	0x3e	62	>
63	? Question Mark	P0 63 (0x3f)	0x3f	63	?
64	@ At Sign	P0 64 (0x40)	0x40	64	@
65	A a Uppercase	P0 65 (0x41)	0x41	65	A
66	B b Uppercase	P0 66 (0x42)	0x42	66	B
67	C c Uppercase	P0 67 (0x43)	0x43	67	C
68	D d Uppercase	P0 68 (0x44)	0x44	68	D
69	E e Uppercase	P0 69 (0x45)	0x45	69	E
70	F f Uppercase	P0 70 (0x46)	0x46	70	F
71	G g Uppercase	P0 71 (0x47)	0x47	71	G
72	H h Uppercase	P0 72 (0x48)	0x48	72	H
73	I i Uppercase	P0 73 (0x49)	0x49	73	I
74	J j Uppercase	P0 74 (0x4a)	0x4a	74	J
75	K k Uppercase	P0 75 (0x4b)	0x4b	75	K
76	L l Uppercase	P0 76 (0x4c)	0x4c	76	L
77	M m Uppercase	P0 77 (0x4d)	0x4d	77	M
78	N n Uppercase	P0 78 (0x4e)	0x4e	78	N
79	O o Uppercase	P0 79 (0x4f)	0x4f	79	O
80	P p Uppercase	P0 80 (0x50)	0x50	80	P
81	Q q Uppercase	P0 81 (0x51)	0x51	81	Q
82	R r Uppercase	P0 82 (0x52)	0x52	82	R
83	S s Uppercase	P0 83 (0x53)	0x53	83	S
84	T t Uppercase	P0 84 (0x54)	0x54	84	T
85	U u Uppercase	P0 85 (0x55)	0x55	85	U
86	V v Uppercase	P0 86 (0x56)	0x56	86	V
87	W w Uppercase	P0 87 (0x57)	0x57	87	W
88	X x Uppercase	P0 88 (0x58)	0x58	88	X
89	Y y Uppercase	P0 89 (0x59)	0x59	89	Y

Figure 4-6 (Part 2 of 8). Code Page P0

display symbols

Font Position	Character	Code Page Code Point	char String	NLchar Value	NCesc Esc Seq
90	Z z Uppercase	P0 90 (0x5a)	0x5a	90	Z
91	[Left Bracket	P0 91 (0x5b)	0x5b	91	[
92	\ Reverse Slash	P0 92 (0x5c)	0x5c	92	\
93] Right Bracket	P0 93 (0x5d)	0x5d	93]
94	^ Circumflex Accent, Up Arrow	P0 94 (0x5e)	0x5e	94	^
95	_ Underline, Low Line	P0 95 (0x5f)	0x5f	95	_
96	` Grave Accent, Left Single Quote	P0 96 (0x60)	0x60	96	`
97	a a Lowercase	P0 97 (0x61)	0x61	97	a
98	b b Lowercase	P0 98 (0x62)	0x62	98	b
99	c c Lowercase	P0 99 (0x63)	0x63	99	c
100	d d Lowercase	P0 100 (0x64)	0x64	100	d
101	e e Lowercase	P0 101 (0x65)	0x65	101	e
102	f f Lowercase	P0 102 (0x66)	0x66	102	f
103	g g Lowercase	P0 103 (0x67)	0x67	103	g
104	h h Lowercase	P0 104 (0x68)	0x68	104	h
105	i i Lowercase	P0 105 (0x69)	0x69	105	i
106	j j Lowercase	P0 106 (0x6a)	0x6a	106	j
107	k k Lowercase	P0 107 (0x6b)	0x6b	107	k
108	l l Lowercase	P0 108 (0x6c)	0x6c	108	l
109	m m Lowercase	P0 109 (0x6d)	0x6d	109	m
110	n n Lowercase	P0 110 (0x6e)	0x6e	110	n
111	o o Lowercase	P0 111 (0x6f)	0x6f	111	o
112	p p Lowercase	P0 112 (0x70)	0x70	112	p
113	q q Lowercase	P0 113 (0x71)	0x71	113	q
114	r r Lowercase	P0 114 (0x72)	0x72	114	r
115	s s Lowercase	P0 115 (0x73)	0x73	115	s
116	t t Lowercase	P0 116 (0x74)	0x74	116	t
117	u u Lowercase	P0 117 (0x75)	0x75	117	u
118	v v Lowercase	P0 118 (0x76)	0x76	118	v
119	w w Lowercase	P0 119 (0x77)	0x77	119	w
120	x x Lowercase	P0 120 (0x78)	0x78	120	x
121	y y Lowercase	P0 121 (0x79)	0x79	121	y
122	z z Lowercase	P0 122 (0x7a)	0x7a	122	z

Figure 4-6 (Part 3 of 8). Code Page P0

display symbols

Font Position	Character	Code Page Code Point	char String	NLchar Value	NCesc Esc Seq
123	{ Left Brace	P0 123 (0x7b)	0x7b	123	{
124	Logical OR	P0 124 (0x7c)	0x7c	124	
125	} Right Brace	P0 125 (0x7d)	0x7d	125	}
126	~ Tilde Accent	P0 126 (0x7e)	0x7e	126	~
127	Δ Del	P0 127 (0x7f)	0x7f	127	Δ
128	Ç c Cedilla Capital	P0 128 (0x80)	0x80	128	\<C,>
129	ü u Umlaut Small	P0 129 (0x81)	0x81	129	\<u">
130	é e Acute Small	P0 130 (0x82)	0x82	130	\<e'>
131	â a Circumflex Small	P0 131 (0x83)	0x83	131	\<a^>
132	ä a Umlaut Small	P0 132 (0x84)	0x84	132	\<a">
133	à a Grave Small	P0 133 (0x85)	0x85	133	\<a'>
134	å a Overcircle Small	P0 134 (0x86)	0x86	134	\<ao>
135	ç c Cedilla Small	P0 135 (0x87)	0x87	135	\<c,>
136	ê e Circumflex Small	P0 136 (0x88)	0x88	136	\<e^>
137	ë e Umlaut Small	P0 137 (0x89)	0x89	137	\<e">
138	è e Grave Small	P0 138 (0x8a)	0x8a	138	\<e'>
139	ï i Umlaut Small	P0 139 (0x8b)	0x8b	139	\<i">
140	î i Circumflex Small	P0 140 (0x8c)	0x8c	140	\<i^>
141	ì i Grave Small	P0 141 (0x8d)	0x8d	141	\<i'>
142	Ä a Umlaut Capital	P0 142 (0x8e)	0x8e	142	\<A">
143	Å a Overcircle Capital	P0 143 (0x8f)	0x8f	143	\<Ao>
144	É e Acute Capital	P0 144 (0x90)	0x90	144	\<E'>
145	æ ae Diphthong Small	P0 145 (0x91)	0x91	145	\<ae>
146	Æ ae Diphthong Capital	P0 146 (0x92)	0x92	146	\<AE>
147	ó o Circumflex Small	P0 147 (0x93)	0x93	147	\<o^>
148	ö o Umlaut Small	P0 148 (0x94)	0x94	148	\<o">
149	ò o Grave Small	P0 149 (0x95)	0x95	149	\<o'>
150	û u Circumflex Small	P0 150 (0x96)	0x96	150	\<u^>
151	ù u Grave Small	P0 151 (0x97)	0x97	151	\<u'>
152	ÿ y Umlaut Small	P0 152 (0x98)	0x98	152	\<y">
153	Ö o Umlaut Capital	P0 153 (0x99)	0x99	153	\<O">
154	Û u Umlaut Capital	P0 154 (0x9a)	0x9a	154	\<U">
155	ø o Slash Small	P0 155 (0x9b)	0x9b	155	\<o/>

Figure 4-6 (Part 4 of 8). Code Page P0

display symbols

Font Position	Character	Code Page Code Point	char String	NLchar Value	NCesc Esc Seq
156	£ English Pound Sign	P0 156 (0x9c)	0x9c	156	\<L=>
157	Ø o Slash Capital	P0 157 (0x9d)	0x9d	157	\<O/>
158	× Multiplication Sign	P0 158 (0x9e)	0x9e	158	\<x>
159	f Florin Sign	P0 159 (0x9f)	0x9f	159	\<f>
160	á a Acute Small	P0 160 (0xa0)	0xa0	160	\<a'>
161	í i Acute Small	P0 161 (0xa1)	0xa1	161	\<i'>
162	ó o Acute Small	P0 162 (0xa2)	0xa2	162	\<o'>
163	ú u Acute Small	P0 163 (0xa3)	0xa3	163	\<u'>
164	ñ n Tilde Small	P0 164 (0xa4)	0xa4	164	\<n~>
165	Ñ n Tilde Capital	P0 165 (0xa5)	0xa5	165	\<N~>
166	♀ Feminine Sign	P0 166 (0xa6)	0xa6	166	\<-a>
167	♂ Masculine Sign	P0 167 (0xa7)	0xa7	167	\<-o>
168	¿ Inverted Question Mark	P0 168 (0xa8)	0xa8	168	\<?>
169	® Registered Trademark	P0 169 (0xa9)	0xa9	169	\<r0>
170	¬ Logical Not	P0 170 (0xaa)	0xaa	170	\<-.>
171	½ One Half	P0 171 (0xab)	0xab	171	\<12>
172	¼ One Quarter	P0 172 (0xac)	0xac	172	\<14>
173	¡ Inverted Exclamation Sign	P0 173 (0xad)	0xad	173	\<!>
174	« Left Angle Quotes	P0 174 (0xae)	0xae	174	\<{>
175	» Right Angle Quotes	P0 175 (0xaf)	0xaf	175	\<}>
176	▤ Quarter Hashed	P0 176 (0xb0)	0xb0	176	\<#1>
177	▥ Half Hashed	P0 177 (0xb1)	0xb1	177	\<#2>
178	▧ Full Hashed	P0 178 (0xb2)	0xb2	178	\<#3>
179	Vertical Bar	P0 179 (0xb3)	0xb3	179	\<S0>
180	‡ Right Side Middle	P0 180 (0xb4)	0xb4	180	\<S6>
181	Á a Acute Capital	P0 181 (0xb5)	0xb5	181	\<A'>
182	Â a Circumflex Capital	P0 182 (0xb6)	0xb6	182	\<A^>
183	Ã a Grave Capital	P0 183 (0xb7)	0xb7	183	\<A'>
184	© Copyright Symbol	P0 184 (0xb8)	0xb8	184	\<c0>
185	‡ Double Right Side Middle	P0 185 (0xb9)	0xb9	185	\<D6>
186	Double Vertical Bar	P0 186 (0xba)	0xba	186	\<D0>
187	↗ Double Upper Right Corner Box	P0 187 (0xbb)	0xbb	187	\<D9>
188	↘ Double Lower Right Corner Box	P0 188 (0xbc)	0xbc	188	\<D3>

Figure 4-6 (Part 5 of 8). Code Page P0

display symbols

Font Position	Character	Code Page Code Point	char String	NLchar Value	NCesc Esc Seq
189	¢ Cent Sign	P0 189 (0xbd)	0xbd	189	\<c/>
190	¥ Yen Sign	P0 190 (0xbe)	0xbe	190	\<Y=>
191	␣ Upper Right Corner Box	P0 191 (0xbf)	0xbf	191	\<S9>
192	␣ Lower Left Corner Box	P0 192 (0xc0)	0xc0	192	\<S1>
193	␣ Bottom Side Middle	P0 193 (0xc1)	0xc1	193	\<S2>
194	␣ Top Side Middle	P0 194 (0xc2)	0xc2	194	\<S8>
195	␣ Left Side Middle	P0 195 (0xc3)	0xc3	195	\<S4>
196	␣ Center Box Bar	P0 196 (0xc4)	0xc4	196	\<S.>
197	⦶ Intersection	P0 197 (0xc5)	0xc5	197	\<S5>
198	ã a Tilde Small	P0 198 (0xc6)	0xc6	198	\<a~>
199	Ã a Tilde Capital	P0 199 (0xc7)	0xc7	199	\<A~>
200	␣ Double Lower Left Corner Box	P0 200 (0xc8)	0xc8	200	\<D1>
201	␣ Double Upper Left Corner Box	P0 201 (0xc9)	0xc9	201	\<D7>
202	␣ Double Bottom Side Middle	P0 202 (0xca)	0xca	202	\<D2>
203	␣ Double Top Side Middle	P0 203 (0xcb)	0xcb	203	\<D8>
204	␣ Double Left Side Middle	P0 204 (0xcc)	0xcc	204	\<D4>
205	␣ Double Center Box Bar	P0 205 (0xcd)	0xcd	205	\<D.>
206	⦶ Double Intersection	P0 206 (0xce)	0xce	206	\<D5>
207	¤ International Currency Symbol	P0 207 (0xcf)	0xcf	207	\<o*>
208	ð eth Icelandic Small	P0 208 (0xd0)	0xd0	208	\<d+>
209	Ð eth Icelandic Capital	P0 209 (0xd1)	0xd1	209	\<D+>
210	Ê e Circumflex Capital	P0 210 (0xd2)	0xd2	210	\<E^>
211	Ë e Umlaut Capital	P0 211 (0xd3)	0xd3	211	\<E">
212	È e Grave Capital	P0 212 (0xd4)	0xd4	212	\<E'>
213	Small i Dotless	P0 213 (0xd5)	0xd5	213	\<i>
214	Í i Acute Capital	P0 214 (0xd6)	0xd6	214	\<I'>
215	Î i Circumflex Capital	P0 215 (0xd7)	0xd7	215	\<I^>
216	Ï i Umlaut Capital	P0 216 (0xd8)	0xd8	216	\<I">
217	␣ Lower Right Corner Box	P0 217 (0xd9)	0xd9	217	\<S3>
218	␣ Upper Left Corner Box	P0 218 (0xda)	0xda	218	\<S7>
219	■ Bright Character Cell	P0 219 (0xdb)	0xdb	219	\
220	■ Bright Character Cell - Lower Half	P0 220 (0xdc)	0xdc	220	\<B2>
221	␣ Broken Vertical Bar	P0 221 (0xdd)	0xdd	221	\<B0>

Figure 4-6 (Part 6 of 8). Code Page P0

display symbols

Font Position	Character	Code Page Code Point	char String	NLchar Value	NCesc Esc Seq
222	Ì i Grave Capital	P0 222 (0xde)	0xde	222	\<I'>
223	■ Bright Character Cell - Upper Half	P0 223 (0xdf)	0xdf	223	\<B8>
224	Ó o Acute Capital	P0 224 (0xe0)	0xe0	224	\<O'>
225	ß s Sharp Small	P0 225 (0xe1)	0xe1	225	\<ss>
226	Ô o Circumflex Capital	P0 226 (0xe2)	0xe2	226	\<O^>
227	Ò o Grave Capital	P0 227 (0xe3)	0xe3	227	\<O>
228	õ o Tilde Small	P0 228 (0xe4)	0xe4	228	\<o~>
229	Õ o Tilde Capital	P0 229 (0xe5)	0xe5	229	\<O~>
230	μ Mu Small, Micro Symbol	P0 230 (0xe6)	0xe6	230	\<&m>
231	þ Thorn Icelandic Small	P0 231 (0xe7)	0xe7	231	\<Ip>
232	Þ Thorn Icelandic Capital	P0 232 (0xe8)	0xe8	232	\<IP>
233	Ú u Acute Capital	P0 233 (0xe9)	0xe9	233	\<U'>
234	Û u Circumflex Capital	P0 234 (0xea)	0xea	234	\<U^>
235	Û u Grave Capital	P0 235 (0xeb)	0xeb	235	\<U>
236	ý y Acute Small	P0 236 (0xec)	0xec	236	\<y'>
237	Ý y Acute Capital	P0 237 (0xed)	0xed	237	\<Y'>
238	— Overbar	P0 238 (0xee)	0xee	238	\<-->
239	' Acute Accent	P0 239 (0xef)	0xef	239	\<_ ' >
240	— Syllable Hyphen	P0 240 (0xf0)	0xf0	240	\<^ - >
241	± Plus Or Minus Sign	P0 241 (0xf1)	0xf1	241	\<+ - >
242	= Double Underscore	P0 242 (0xf2)	0xf2	242	\<_ _ >
243	¾ Three Fourths	P0 243 (0xf3)	0xf3	243	\<34>
244	¶ Paragraph Symbol	P0 244 (0xf4)	0xf4	244	\< P>
245	§ Section Symbol	P0 245 (0xf5)	0xf5	245	\< S>
246	÷ Division Sign	P0 246 (0xf6)	0xf6	246	\<: - >
247	¸ Cedilla Accent	P0 247 (0xf7)	0xf7	247	\<_ , >
248	° Degree Symbol, Overcircle Accent	P0 248 (0xf8)	0xf8	248	\<o>
249	¨ Umlaut Accent	P0 249 (0xf9)	0xf9	249	\<_ " >
250	· Middle Dot, Product Dot	P0 250 (0xfa)	0xfa	250	\<_ . >
251	¹ Superscript 1	P0 251 (0xfb)	0xfb	251	\<^ 1 >
252	³ Superscript 3	P0 252 (0xfc)	0xfc	252	\<^ 3 >

Figure 4-6 (Part 7 of 8). Code Page P0

display symbols

Font Position	Character	Code Page Code Point	char String	NLchar Value	NCesc Esc Seq
253	² Superscript 2	P0 253 (0xfd)	0xfd	253	\<^2>
254	▮ Vertical Solid Rectangle	P0 254 (0xfe)	0xfe	254	\<[]>
255	Required Space	P0 255 (0xff)	0xff	255	\<##>

Figure 4-6 (Part 8 of 8). Code Page P0

Font Position	Character	Code Page Code Point	char String	NLchar Value	NCesc Esc Seq
256	• Spanish Middle Dot	P1 32 (0x20)	0x1fa0	288	\<_.>
257	☺ Smiling Face	P1 33 (0x21)	0x1fa1	289	\<:)>
258	☹ Dark Smiling Face	P1 34 (0x22)	0x1fa2	290	\<(:>
259	♥ Heart	P1 35 (0x23)	0x1fa3	291	\<SH>
260	♦ Diamond	P1 36 (0x24)	0x1fa4	292	\<SD>
261	♣ Club	P1 37 (0x25)	0x1fa5	293	\<SC>
262	♠ Spade	P1 38 (0x26)	0x1fa6	294	\<SS>
263	• Bullet	P1 39 (0x27)	0x1fa7	295	\<@>
264	◼ Reverse Video Bullet	P1 40 (0x28)	0x1fa8	296	\<@#>
265	○ Circle	P1 41 (0x29)	0x1fa9	297	\<O>
266	◉ Reverse Video Circle	P1 42 (0x2a)	0x1faa	298	\<O#>
267	♂ Male Symbol	P1 43 (0x2b)	0x1fab	299	\<o%>
268	♀ Female Symbol	P1 44 (0x2c)	0x1fac	300	\<o+>
269	♪ Eighth Note	P1 45 (0x2d)	0x1fad	301	\<d~>
270	♫ Sixteenth Note	P1 46 (0x2e)	0x1fae	302	\<d=>
271	☼ Sun	P1 47 (0x2f)	0x1faf	303	\<*>
272	▶ Right Solid Triangle	P1 48 (0x30)	0x1fb0	304	\<#}>
273	◀ Left Solid Triangle	P1 49 (0x31)	0x1fb1	305	\<{#>
274	↑ Bidirectional Vertical Arrow	P1 50 (0x32)	0x1fb2	306	\<^v>
275	!! Double Exclamation Point	P1 51 (0x33)	0x1fb3	307	\<!!>
276	¶ Paragraph Symbol	P1 52 (0x34)	0x1fb4	308	\< P>
277	§ Section symbol	P1 53 (0x35)	0x1fb5	309	\< S>
278	▬ Horizontal Solid Rectangle	P1 54 (0x36)	0x1fb6	310	\<#]>
279	↕ Underlined Bidirectional Vertical Arrow	P1 55 (0x37)	0x1fb7	311	\<- >

Figure 4-7 (Part 1 of 8). Code Page P1

display symbols

Font Position	Character	Code Page Code Point	char String	NLchar Value	NCesc Esc Seq
280	↑ Cursor Up	P1 56 (0x38)	0x1fb8	312	\< ^>
281	↓ Cursor Down	P1 57 (0x39)	0x1fb9	313	\< v>
282	→ Cursor Right	P1 58 (0x3a)	0x1fba	314	\<-}>
283	← Cursor Left	P1 59 (0x3b)	0x1fbb	315	\<{->
284	⌞ Diagonally Flipped Logical Not	P1 60 (0x3c)	0x1fbc	316	\<'>
285	↔ Bidirectional Horizontal Arrow	P1 61 (0x3c)	0x1fbd	317	\<(>
286	▲ Solid Upward Triangle	P1 62 (0x3e)	0x1fbe	318	\<#^>
287	▼ Solid Downward Triangle	P1 63 (0x3f)	0x1fbf	319	\<#v>
288	ã a Tilde Small	P1 64 (0x40)	0x1fc0	320	\<a~>
289	ß s Sharp Small	P1 65 (0x41)	0x1fc1	321	\<ss>
290	Â a Circumflex Capital	P1 66 (0x42)	0x1fc2	322	\<A^>
291	À a Grave Capital	P1 67 (0x43)	0x1fc3	323	\<A'>
292	Á a Acute Capital	P1 68 (0x44)	0x1fc4	324	\<A'>
293	Ã a Tilde Capital	P1 69 (0x45)	0x1fc5	325	\<A~>
294	ø o Slash Small	P1 70 (0x46)	0x1fc6	326	\<o/>
295	Ê e Circumflex Capital	P1 71 (0x47)	0x1fc7	327	\<E^>
296	Ë e Umlaut Capital	P1 72 (0x48)	0x1fc8	328	\<E">
297	Ë e Grave Capital	P1 73 (0x49)	0x1fc9	329	\<E'>
298	Í i Acute Capital	P1 74 (0x4a)	0x1fca	330	\<I'>
299	Î i Circumflex Capital	P1 75 (0x4b)	0x1fcb	331	\<I^>
300	Ï i Umlaut Capital	P1 76 (0x4c)	0x1fcc	332	\<I">
301	Ì i Grave Capital	P1 77 (0x4d)	0x1fcd	333	\<I'>
302	Ø Slashed o Capital	P1 78 (0x4e)	0x1fce	334	\<O/>
303	ð eth Icelandic Small	P1 79 (0x4f)	0x1fcf	335	\<d+>
304	ý y Acute Small	P1 80 (0x50)	0x1fd0	336	\<y'>
305	þ Thorn Icelandic Small	P1 81 (0x51)	0x1fd1	337	\<Ip>
306	¸ Cedilla Accent	P1 82 (0x52)	0x1fd2	338	\<-,>
307	¤ International Currency Symbol	P1 83 (0x53)	0x1fd3	339	\<o*>
308	Ð eth Icelandic Capital	P1 84 (0x54)	0x1fd4	340	\<D+>
309	Ý y Acute Capital	P1 85 (0x55)	0x1fd5	341	\<Y'>
310	Þ Thorn Icelandic Capital	P1 86 (0x56)	0x1fd6	342	\<IP>
311	® Registered Trademark Symbol	P1 87 (0x57)	0x1fd7	343	\<r0>
312	¾ Three Quarters	P1 88 (0x58)	0x1fd8	344	\<34>

Figure 4-7 (Part 2 of 8). Code Page P1

display symbols

Font Position	Character	Code Page Code Point	char String	NLchar Value	NCesc Esc Seq
313	—	P1 89 (0x59)	0x1fd9	345	\<-->
314	¨	P1 90 (0x5a)	0x1fda	346	\<_>
315	'	P1 91 (0x5b)	0x1fdb	347	\<_>
316	=	P1 92 (0x5c)	0x1fdc	348	\<_>
317	o	P1 93 (0x5d)	0x1fdd	349	\<o~>
318	i	P1 94 (0x5e)	0x1fde	350	\<i>
319	Ô	P1 95 (0x5f)	0x1fdf	351	\<O^>
320	Ò	P1 96 (0x60)	0x1fe0	352	\<O'>
321	Ó	P1 97 (0x61)	0x1fe1	353	\<O'>
322	Õ	P1 98 (0x62)	0x1fe2	354	\<O~>
323	³	P1 99 (0x63)	0x1fe3	355	\<^3>
324	Û	P1 100 (0x64)	0x1fe4	356	\<U^>
325	Û	P1 101 (0x65)	0x1fe5	357	\<U'>
326	Ů	P1 102 (0x66)	0x1fe6	358	\<U'>
327	ą	P1 103 (0x67)	0x1fe7	359	\<a,>
328	ě	P1 104 (0x68)	0x1fe8	360	\<ev>
329	č	P1 105 (0x69)	0x1fe9	361	\<cv>
330	ć	P1 106 (0x6a)	0x1fea	362	\<c'>
331	ę	P1 107 (0x6b)	0x1feb	363	\<e,>
332	ü	P1 108 (0x6c)	0x1fec	364	\<uo>
333	ď	P1 109 (0x6d)	0x1fed	365	\<dv>
334	ĺ	P1 110 (0x6e)	0x1fee	366	\<l'>
335	Ą	P1 111 (0x6f)	0x1fef	367	\<A,>
336	Ě	P1 112 (0x70)	0x1ff0	368	\<Ev>
337	Č	P1 113 (0x71)	0x1ff1	369	\<Cv>
338	Ć	P1 114 (0x72)	0x1ff2	370	\<C'>
339	ˆ	P1 115 (0x73)	0x1ff3	371	\<_v>
340	Ę	P1 116 (0x74)	0x1ff4	372	\<E,>
341	Ů	P1 117 (0x75)	0x1ff5	373	\<Uo>
342	Ď	P1 118 (0x76)	0x1ff6	374	\<Dv>
343	Ĺ	P1 119 (0x77)	0x1ff7	375	\<L'>
344	ĺ	P1 120 (0x78)	0x1ff8	376	\<l'v>
345	ň	P1 121 (0x79)	0x1ff9	377	\<nv>

Figure 4-7 (Part 3 of 8). Code Page P1

display symbols

Font Position	Character	Code Page Code Point	char String	NLchar Value	NCesc Esc Seq
346	đ	P1 122 (0x7a)	0x1ffa	378	\<d->
347	ŗ	P1 123 (0x7b)	0x1ffb	379	\<rv>
348	ŝ	P1 124 (0x7c)	0x1ffc	380	\<s'>
349	◌̊	P1 125 (0x7d)	0x1ffd	381	\<_o>
350	ł	P1 126 (0x7e)	0x1ffe	382	\<l->
351	ń	P1 127 (0x7f)	0x1fff	383	\<n'>
352	ŝ	P1 128 (0x80)	0x1e80	384	\<sv>
353	Ł	P1 129 (0x81)	0x1e81	385	\<Lv>
354	Ń	P1 130 (0x82)	0x1e82	386	\<Nv>
355	Ŕ	P1 131 (0x83)	0x1e83	387	\<Rv>
356	Ŝ	P1 132 (0x84)	0x1e84	388	\<S'>
357	◌̇	P1 133 (0x85)	0x1e85	389	\<.>
358	ż	P1 134 (0x86)	0x1e86	390	\<z.>
359	˛	P1 135 (0x87)	0x1e87	391	\<.,>
360	Ż	P1 136 (0x88)	0x1e88	392	\<Z.>
361	ż	P1 137 (0x89)	0x1e89	393	\<zv>
362	Ź	P1 138 (0x8a)	0x1e8a	394	\<Z'>
363	Ź	P1 139 (0x8b)	0x1e8b	395	\<Zv>
364	Ž	P1 140 (0x8c)	0x1e8c	396	\<Z'>
365	Ł	P1 141 (0x8d)	0x1e8d	397	\<L->
366	Ń	P1 142 (0x8e)	0x1e8e	398	\<N'>
367	Ŝ	P1 143 (0x8f)	0x1e8f	399	\<Sv>
368	ŗ	P1 144 (0x90)	0x1e90	400	\<tv>
369	ŕ	P1 145 (0x91)	0x1e91	401	\<r'>
370	◌̈	P1 146 (0x92)	0x1e92	402	\<o=>
371	◌̈	P1 147 (0x93)	0x1e93	403	\<u=>
372	Ť	P1 148 (0x94)	0x1e94	404	\<Tv>
373	Ŗ	P1 149 (0x95)	0x1e95	405	\<R'>
374	Ŏ	P1 150 (0x96)	0x1e96	406	\<O=>
375	Ũ	P1 151 (0x97)	0x1e97	407	\<U=>
376	ă	P1 152 (0x98)	0x1e98	408	\<au>
377	ġ	P1 153 (0x99)	0x1e99	409	\<gu>
378	İ	P1 154 (0x9a)	0x1e9a	410	\<I.>

Figure 4-7 (Part 4 of 8). Code Page P1

display symbols

Font Position	Character	Code Page Code Point	char String	NLchar Value	NCesc Esc Seq
379	À	P1 155 (0x9b)	0x1e9b	411	\<Au>
380	Á	P1 156 (0x9c)	0x1e9c	412	\<Gu>
381	˘	P1 157 (0x9d)	0x1e9d	413	\<-u>
382	¨	P1 158 (0x9e)	0x1e9e	414	\<-=>
383	¸	P1 159 (0x9f)	0x1e9f	415	\<s,>
384	ℓ	P1 160 (0xa0)	0x1ea0	416	\<l>
385	ˆn	P1 161 (0xa1)	0x1ea1	417	\<,n>
386	Š	P1 162 (0xa2)	0x1ea2	418	\<S,>
387	ˉ	P1 163 (0xa3)	0x1ea3	419	\<- ->
388	ţ	P1 164 (0xa4)	0x1ea4	420	\<t,>
389	Ț	P1 165 (0xa5)	0x1ea5	421	\<T,>
390	ā	P1 166 (0xa6)	0x1ea6	422	\<a ->
391	Ā	P1 167 (0xa7)	0x1ea7	423	\<A ->
392	ĉ	P1 168 (0xa8)	0x1ea8	424	\<c^>
393	Ĉ	P1 169 (0xa9)	0x1ea9	425	\<C^>
394	ˆ	P1 170 (0xaa)	0x1eaa	426	\<\\>
395	ċ	P1 171 (0xab)	0x1eab	427	\<c.>
396	Ċ	P1 172 (0xac)	0x1eac	428	\<C.>
397	ė	P1 173 (0xad)	0x1ead	429	\<e.>
398	Ė	P1 174 (0xae)	0x1eae	430	\<E.>
399	ē	P1 175 (0xaf)	0x1eaf	431	\<e ->
400	Ē	P1 176 (0xb0)	0x1eb0	432	\<E ->
401	ġ	P1 177 (0xb1)	0x1eb1	433	\<g'>
402	ġ	P1 178 (0xb2)	0x1eb2	434	\<g^>
403	Ġ	P1 179 (0xb3)	0x1eb3	435	\<G^>
404	ġ	P1 180 (0xb4)	0x1eb4	436	\<g.>
405	Ġ	P1 181 (0xb5)	0x1eb5	437	\<G.>
406	Ġ	P1 182 (0xb6)	0x1eb6	438	\<G,>
407	ĥ	P1 183 (0xb7)	0x1eb7	439	\<h^>
408	Ĥ	P1 184 (0xb8)	0x1eb8	440	\<H^>
409	ħ	P1 185 (0xb9)	0x1eb9	441	\<h ->
410	Ĥ	P1 186 (0xba)	0x1eba	442	\<H ->
411	ı	P1 187 (0xbb)	0x1ebb	443	\<i~>

Figure 4-7 (Part 5 of 8). Code Page P1

Font Position	Character	Code Page Code Point	char String	NLchar Value	NCesc Esc Seq
412	İ	P1 188 (0xbc)	0x1ebc	444	\<I~>
413	ı	P1 189 (0xbd)	0x1ebd	445	\<i->
414	Ī	P1 190 (0xbe)	0x1ebe	446	\<I->
415	i	P1 191 (0xbf)	0x1ebf	447	\<i,>
416	Ĭ	P1 192 (0xc0)	0x1ec0	448	\<I,>
417	ij	P1 193 (0xc1)	0x1ec1	449	\<ij>
418	IJ	P1 194 (0xc2)	0x1ec2	450	\<IJ>
419	ĵ	P1 195 (0xc3)	0x1ec3	451	\<j^>
420	J	P1 196 (0xc4)	0x1ec4	452	\<J^>
421	ķ	P1 197 (0xc5)	0x1ec5	453	\<k,>
422	K	P1 198 (0xc6)	0x1ec6	454	\<K,>
423	ƙ	P1 199 (0xc7)	0x1ec7	455	\<k>
424	ļ	P1 200 (0xc8)	0x1ec8	456	\<l,>
425	L	P1 201 (0xc9)	0x1ec9	457	\<L,>
426	ł	P1 202 (0xca)	0x1eca	458	\<l.>
427	Ł	P1 203 (0xcb)	0x1ecb	459	\<L.>
428	ņ	P1 204 (0xcc)	0x1ecc	460	\<n,>
429	N	P1 205 (0xcd)	0x1ecd	461	\<N,>
430	ŋ	P1 206 (0xce)	0x1ece	462	\<nj>
431	Ņ	P1 207 (0xcf)	0x1ecf	463	\<Nj>
432	ō	P1 208 (0xd0)	0x1ed0	464	\<o->
433	Ō	P1 209 (0xd1)	0x1ed1	465	\<O->
434	œ	P1 210 (0xd2)	0x1ed2	466	\<oe>
435	Œ	P1 211 (0xd3)	0x1ed3	467	\<OE>
436	ŗ	P1 212 (0xd4)	0x1ed4	468	\<r,>
437	R	P1 213 (0xd5)	0x1ed5	469	\<R,>
438	ŝ	P1 214 (0xd6)	0x1ed6	470	\<s^>
439	Ŝ	P1 215 (0xd7)	0x1ed7	471	\<S^>
440	ţ	P1 216 (0xd8)	0x1ed8	472	\<t->
441	T	P1 217 (0xd9)	0x1ed9	473	\<T->
442	ũ	P1 218 (0xda)	0x1eda	474	\<u~>
443	Ũ	P1 219 (0xdb)	0x1edb	475	\<U~>
444	ū	P1 220 (0xdc)	0x1edc	476	\<uu>

Figure 4-7 (Part 6 of 8). Code Page P1

display symbols

Font Position	Character	Code Page Code Point	char String	NLchar Value	NCesc Esc Seq
445	Ů u Breve Capital	P1 221 (0xdd)	0x1edd	477	\<Uu>
446	ū u Macron Small	P1 222 (0xde)	0x1ede	478	\<u->
447	Ū u Macron Capital	P1 223 (0xdf)	0x1edf	479	\<U->
448	ȩ u Ogonek Small	P1 224 (0xe0)	0x1ee0	480	\<u,>
449	Ů u Ogonek Capital	P1 225 (0xe1)	0x1ee1	481	\<U,>
450	ŵ w Circumflex Small	P1 226 (0xe2)	0x1ee2	482	\<w^>
451	Ŵ w Circumflex Capital	P1 227 (0xe3)	0x1ee3	483	\<W^>
452	ÿ y Circumflex Small	P1 228 (0xe4)	0x1ee4	484	\<y^>
453	Ÿ y Circumflex Capital	P1 229 (0xe5)	0x1ee5	485	\<Y^>
454	ÿ y Umlaut Capital	P1 230 (0xe6)	0x1ee6	486	\<Y">
455	© Copyright Symbol	P1 231 (0xe7)	0x1ee7	487	\<c0>
456	¹ Superscript One	P1 232 (0xe8)	0x1ee8	488	\<^1>
457	™ Trademark Symbol	P1 233 (0xe9)	0x1ee9	489	\<tm>
458	¹ / ₈ One Eighth	P1 234 (0xea)	0x1eea	490	\<18>
459	³ / ₈ Three Eighths	P1 235 (0xeb)	0x1eeb	491	\<38>
460	⁵ / ₈ Five Eighths	P1 236 (0xec)	0xleec	492	\<58>
461	⁷ / ₈ Seven Eighths	P1 237 (0xed)	0xleed	493	\<78>
462	× Multiplication Sign	P1 238 (0xee)	0xleee	494	\<x>
463	' Right Single Quotation Mark	P1 239 (0xef)	0xleef	495	\<'>
464	" Left Double Quotation Mark	P1 240 (0xf0)	0xlef0	496	\<">
465	" Right Double Quotation Mark	P1 241 (0xf1)	0xlef1	497	\<' '>
466	= Equal Sign Superscript	P1 242 (0xf2)	0xlef2	498	\<^=>
467	- Minus Sign Superscript	P1 243 (0xf3)	0xlef3	499	\<^->
468	+ Plus Sign Superscript	P1 244 (0xf4)	0xlef4	500	\<^+>
469	∞ Infinity symbol Superscript	P1 245 (0xf5)	0xlef5	501	\<8^>
470	π Pi Symbol Superscript	P1 246 (0xf6)	0xlef6	502	\<^p>
471	Δ Delta Symbol Superscript	P1 247 (0xf7)	0xlef7	503	\<^d>
472	→ Right Arrow Superscript	P1 248 (0xf8)	0xlef8	504	\<^}>
473	/ Slash Superscript	P1 249 (0xf9)	0xlef9	505	\<^/>
474	† Dagger	P1 250 (0xfa)	0xlefa	506	\< +>
475	< Left Angle Superscript	P1 251 (0xfb)	0xlefb	507	\<^[>
476	> Right Angle Superscript	P1 252 (0xfc)	0xlefc	508	\<^]>
477	℞ Prescription Symbol	P1 253 (0xfd)	0xlefd	509	\<Rx>

Figure 4-7 (Part 7 of 8). Code Page P1

display symbols

Font Position	Character	Code Page Code Point	char String	NLchar Value	NCesc Esc Seq
478	̸ 'Is Not An Element' Symbol	P1 254 (0xfe)	0x1efe	510	\<e/>
479	∴ 'Therefore' Symbol	P1 255 (0xff)	0x1eff	511	\<.:>

Figure 4-7 (Part 8 of 8). Code Page P1

Font Position	Character	Code Page Code Point	char String	NLchar Value	NCesc Esc Seq
480	↗ Increase	P2 32 (0x20)	0x1da0	544	\</^>
481	↘ Decrease	P2 33 (0x21)	0x1da1	545	\<"/v>
482	‡ Double Dagger	P2 34 (0x22)	0x1da2	546	\<+>
483	≠ Not Equal Symbol	P2 35 (0x23)	0x1da3	547	\</=>
484	✓ OR Symbol	P2 36 (0x24)	0x1da4	548	\<v>
485	^ AND Symbol	P2 37 (0x25)	0x1da5	549	\<^>
486	Parallel	P2 38 (0x26)	0x1da6	550	\< >
487	∠ Angle Symbol	P2 39 (0x27)	0x1da7	551	\</_>
488	< Left Angle Bracket	P2 40 (0x28)	0x1da8	552	\<{>
489	> Right Angle Bracket	P2 41 (0x29)	0x1da9	553	\<}>
490	± Minus Or Plus Sign	P2 42 (0x2a)	0x1daa	554	\<-+>
491	◻ Lozenge	P2 43 (0x2b)	0x1dab	555	\<{>
492	' Minutes Symbol	P2 44 (0x2c)	0x1dac	556	\<'>
493	∫ Integral Symbol	P2 45 (0x2d)	0x1dad	557	\<S>
494	∪ Union	P2 46 (0x2e)	0x1dae	558	\<u>
495	⊂ 'Is Included In' Symbol	P2 47 (0x2f)	0x1daf	559	\<(->
496	⊃ 'Includes' Symbol	P2 48 (0x30)	0x1db0	560	\<(-)>
497	⊕ Circle Plus, Closed Sum	P2 49 (0x31)	0x1db1	561	\<0+>
498	⊥ Right Angle Symbol	P2 50 (0x32)	0x1db2	562	\<L>
499	⊗ Circle Multiply	P2 51 (0x33)	0x1db3	563	\<0x>
500	" Seconds Symbol	P2 52 (0x34)	0x1db4	564	\<">
501	= Double Overline	P2 53 (0x35)	0x1db5	565	\<=>
502	ψ Psi Small	P2 54 (0x36)	0x1db6	566	\<&y>
503	ε Epsilon Small	P2 55 (0x37)	0x1db7	567	\<&e>
504	λ Lambda Small	P2 56 (0x38)	0x1db8	568	\<&l>

Figure 4-8 (Part 1 of 5). Code Page P2

display symbols

Font Position	Character	Code Page Code Point	char String	NLchar Value	NCesc Esc Seq
505	η	P2 57 (0x39)	0x1db9	569	\<&h>
506	ι	P2 58 (0x3a)	0x1dba	570	\<&i>
507	(P2 59 (0x3b)	0x1dbb	571	\<^(>
508)	P2 60 (0x3c)	0x1dbc	572	\<v(>
509	‰	P2 61 (0x3d)	0x1dbd	573	\<%>
510	θ	P2 62 (0x3e)	0x1dbe	574	\<&>
511	κ	P2 63 (0x3f)	0x1dbf	575	\<&k>
512	ω	P2 64 (0x40)	0x1dc0	576	\<&w>
513	ν	P2 65 (0x41)	0x1dc1	577	\<&n>
514	ο	P2 66 (0x42)	0x1dc2	578	\<&o>
515	ρ	P2 67 (0x43)	0x1dc3	579	\<&r>
516	γ	P2 68 (0x44)	0x1dc4	580	\<&g>
517	θ	P2 69 (0x45)	0x1dc5	581	\<&q>
518)	P2 70 (0x46)	0x1dc6	582	\<^)>
519)	P2 71 (0x47)	0x1dc7	583	\<v)>
520	≅	P2 72 (0x48)	0x1dc8	584	\<~=>
521	ξ	P2 73 (0x49)	0x1dc9	585	\<&x>
522	χ	P2 74 (0x4a)	0x1dca	586	\<&c>
523	υ	P2 75 (0x4b)	0x1dcb	587	\<&u>
524	ζ	P2 76 (0x4c)	0x1dcc	588	\<&z>
525]	P2 77 (0x4d)	0x1dcd	589	\< '>
526	[P2 78 (0x4e)	0x1dce	590	\<' >
527	₀	P2 79 (0x4f)	0x1dcf	591	\<v0>
528	₁	P2 80 (0x50)	0x1dd0	592	\<v1>
529	₂	P2 81 (0x51)	0x1dd1	593	\<v2>
530	₃	P2 82 (0x52)	0x1dd2	594	\<v3>
531	₄	P2 83 (0x53)	0x1dd3	595	\<v4>
532	₅	P2 84 (0x54)	0x1dd4	596	\<v5>
533	₆	P2 85 (0x55)	0x1dd5	597	\<v6>
534	₇	P2 86 (0x56)	0x1dd6	598	\<v7>
535	₈	P2 87 (0x57)	0x1dd7	599	\<v8>
536	₉	P2 88 (0x58)	0x1dd8	600	\<v9>
537	⊥	P2 89 (0x59)	0x1dd9	601	\< ->

Figure 4-8 (Part 2 of 5). Code Page P2

display symbols

Font Position	Character	Code Page Code Point	char String	NLchar Value	NCesc Esc Seq
538	⊕ Total Symbol	P2 90 (0x5a)	0x1dda	602	\<-^>
539	Ψ Psi Capital	P2 91 (0x5b)	0x1ddb	603	\<&Y>
540	Π Pi Capital	P2 92 (0x5c)	0x1ddc	604	\<&P>
541	Λ Lambda Capital	P2 93 (0x5d)	0x1ddd	605	\<&L>
542	● Bottle Symbol	P2 94 (0x5e)	0x1dde	606	\<db>
543	␣ Substitute Blank	P2 95 (0x5f)	0x1ddf	607	\
544	∂ Partial Differential Symbol	P2 96 (0x60)	0x1de0	608	\<d>
545	~ Sine Symbol	P2 97 (0x61)	0x1de1	609	\<~>
546	□ Open Square	P2 98 (0x62)	0x1de2	610	\<[->
547	■ Solid Square	P2 99 (0x63)	0x1de3	611	\<[#>
548	◻ Slash Square	P2 100 (0x64)	0x1de4	612	\<[/>
549	∑ Upper Summation Section	P2 101 (0x65)	0x1de5	613	\<^S>
550	∑ Lower Summation Section	P2 102 (0x66)	0x1de6	614	\<vS>
551	Ξ Xi Capital	P2 103 (0x67)	0x1de7	615	\<&X>
552	∝ 'Proportional To' Symbol	P2 104 (0x68)	0x1de8	616	\<o(>
553	Δ Delta Capital	P2 105 (0x69)	0x1de9	617	\<&D>
554	Υ Upsilon Capital	P2 106 (0x6a)	0x1dea	618	\<&U>
555	≈ 'Approximately Equal To' Symbol	P2 107 (0x6b)	0x1deb	619	\<~->
556	~ Cycle Symbol, 'Equivalent To' Symbol	P2 108 (0x6c)	0x1dec	620	\<~>
557	⁰ Zero Superscript	P2 109 (0x6d)	0x1ded	621	\<^0>
558	⁴ Four Superscript	P2 110 (0x6e)	0x1dee	622	\<^4>
559	⁵ Five Superscript	P2 111 (0x6f)	0x1def	623	\<^5>
560	⁶ Six Superscript	P2 112 (0x70)	0x1df0	624	\<^6>
561	⁷ Seven Superscript	P2 113 (0x71)	0x1df1	625	\<^7>
562	⁸ Eight Superscript	P2 114 (0x72)	0x1df2	626	\<^8>
563	⁹ Nine Superscript	P2 115 (0x73)	0x1df3	627	\<^9>
564	Ø Zero Slash	P2 116 (0x74)	0x1df4	628	\<0/>
565	℞ Paseto Sign	P2 117 (0x75)	0x1df5	629	\<Pt>
566	⌋ Flipped Logical Not	P2 118 (0x76)	0x1df6	630	\<.->
567	⌋ Right Side Middle - Double Horizontal	P2 119 (0x77)	0x1df7	631	\<H6>
568	⌋ Right Side Middle - Double Vertical	P2 120 (0x78)	0x1df8	632	\<V6>
569	⌋ Upper Right Corner - Double Vertical	P2 121 (0x79)	0x1df9	633	\<V9>
570	⌋ Upper Right Corner - Double Hor.	P2 122 (0x7a)	0x1dfa	634	\<H9>

Figure 4-8 (Part 3 of 5). Code Page P2

display symbols

Font Position	Character	Code Page Code Point	char String	NLchar Value	NCesc Esc Seq
571	┘	P2 123 (0x7b)	0x1dfb	635	\<V3>
572	┘	P2 124 (0x7c)	0x1dfc	636	\<H3>
573	┘	P2 125 (0x7d)	0x1dfd	637	\<H4>
574	┘	P2 126 (0x7e)	0x1dfe	638	\<V4>
575	┘	P2 127 (0x7f)	0x1dff	639	\<H2>
576	┘	P2 128 (0x80)	0x1c80	640	\<V2>
577	┘	P2 129 (0x81)	0x1c81	641	\<H8>
578	┘	P2 130 (0x82)	0x1c82	642	\<V1>
579	┘	P2 131 (0x83)	0x1c83	643	\<H1>
580	┘	P2 132 (0x84)	0x1c84	644	\<H7>
581	┘	P2 133 (0x85)	0x1c85	645	\<V7>
582	┘	P2 134 (0x86)	0x1c86	646	\<V5>
583	┘	P2 135 (0x87)	0x1c87	647	\<H5>
584	■	P2 136 (0x88)	0x1c88	648	\<B4>
585	■	P2 137 (0x89)	0x3c89	649	\<B6>
586	α	P2 138 (0x8a)	0x1c8a	650	\<&a>
587	β	P2 139 (0x8b)	0x1c8b	651	\<&b>
588	Γ	P2 140 (0x8c)	0x1c8c	652	\<&G>
589	π	P2 141 (0x8d)	0x1c8d	653	\<&p>
590	Σ	P2 142 (0x8e)	0x1c8e	654	\<&S>
591	σ	P2 143 (0x8f)	0x1c8f	655	\<&s>
592	τ	P2 144 (0x90)	0x1c90	656	\<&t>
593	Φ	P2 145 (0x91)	0x1c91	657	\<&F>
594	Θ	P2 146 (0x92)	0x1c92	658	\<&Q>
595	Ω	P2 147 (0x93)	0x1c93	659	\<&W>
596	δ	P2 148 (0x94)	0x1c94	660	\<&d>
597	∞	P2 149 (0x95)	0x1c95	661	\<8>
598	φ	P2 150 (0x96)	0x1c96	662	\<&f>
599	∈	P2 151 (0x97)	0x1c97	663	\<e>
600	∩	P2 152 (0x98)	0x1c98	664	\<n>
601	≡	P2 153 (0x99)	0x1c99	665	\<==>
602	≥	P2 154 (0x9a)	0x1c9a	666	\<}>=
603	≤	P2 155 (0x9b)	0x1c9b	667	\<{>=

Figure 4-8 (Part 4 of 5). Code Page P2

Font Position	Character	Code Page Code Point	char String	NLchar Value	NCesc Esc Seq
604	⌈	P2 156 (0x9c)	0x1c9c	668	\<^ I>
605	⌋	P2 157 (0x9d)	0x1c9d	669	\<v I>
606	≈	P2 158 (0x9e)	0x1c9e	670	\<= ~>
607	∘	P2 159 (0x9f)	0x1c9f	671	\<# o>
608	√	P2 160 (0xa0)	0x1ca0	672	\<v ->
609	⌘	P2 161 (0xa1)	0x1ca1	673	\<V8>
610	ⁿ	P2 162 (0xa2)	0x1ca2	674	\<^ n>
611		P2 163 (0xa3)	0x1ca3	675	\<0 ->
612	⌞	P2 164 (0xa4)	0x1ca4	676	\<-)>
613	⌜	P2 165 (0xa5)	0x1ca5	677	\<- u>
614	⌞	P2 166 (0xa6)	0x1ca6	678	\<I v>
615	⌞	P2 167 (0xa7)	0x1ca7	679	\<' v>
616	∅	P2 168 (0xa8)	0x1ca8	680	\<- =>

Figure 4-8 (Part 5 of 5). Code Page P2

File

/usr/pub/ibmcharset Contains RT character set.

Related Information

In this book: “conv” on page 2-105, “NLchar” on page 2-514, “NLescstr, NLunescstr, NLflatstr” on page 2-516, “fonts” on page 3-79, “data stream” on page 4-5, “hft” on page 5-39, and “keyboard” on page 5-97.

Keyboard Description and Character Reference.

ebcdic

ebcdic

Purpose

Maps the EBCDIC character set.

Synopsis

`cat /usr/pub/ebcdic`

Description

In the following table, columns correspond to the high-order hexadecimal digits, and rows correspond to low-order hexadecimal digits. The cells contain equivalent hexadecimal ASCII values, the symbols, and mnemonics common to EBCDIC and ASCII. Exceptions are flagged in the following table by (1) through (8):

	0_	1_	2_	3_	4_	5_	6_	7_
_0	00 nul	10 dle	80 ds	90	20 sp	26 &	2D -	BA
_1	01 soh	11 dc1	81 sos	91	A0 sp	A9	2F /	BB
_2	02 stx	12 dc2	82 fs	16 syn	A1	AA	B2	BC
_3	03 etx	13 (1)	83	93	A2	AB	B3	BD
_4	9C pf	9D res	84 byp	94 pn	A3	AC	B4	BE
_5	09 ht	85 nl	0A lf	95 rs	A4	AD	B5	BF
_6	86 lc	08 bs	17 etb	96 uc	A5	AE	B6	C0
_7	7F del	87 il	1B esc	04 eot	A6	AF	B7	C1
_8	97	18 can	88	98	A7	B0	B8	C2
_9	8D	19 em	89	99	A8	B1	B9	60 ' ,
_A	8E smm	92 cc	8A sm	9A	D5	21 !	CB	3A :
_B	0B vt	8F cu1	8B cu2	9B cu3	2E .	24 \$	2C ,	23 #
_C	0C ff	1C (2)	8C	14 dc4	3C <	2A *	25 %	40 •
_D	0D cr	1D (3)	05 enq	15 nak	28 (29)	5F -	27 ' ,
_E	0E so	1E (4)	06 ack	9E	2B +	3B ;	3E >	3D =
_F	0F si	1F (5)	07 bel	1A sub	7C l(6)	7E ~(7)	3F ?	22 " ,

Figure 4-9 (Part 1 of 2). EBCDIC Character Set

8_	9_	A_	B_	C_	D_	E_	F_
_0 C3	CA	D1	D8	7B {	7D }	5C \	30 0
_1 61 a	6A j	E5	D9	41 A	4A J	9F	31 1
_2 62 b	6B k	73 s	DA	42 B	4B K	53	32 2
_3 63 c	6C l	74 t	DB	43 C	4C L	54	33 3
_4 64 d	6D m	75 u	DC	44 D	4D M	55	34 4
_5 65 e	6E n	76 v	DD	45 E	4E N	56	35 5
_6 66 f	6F o	77 w	DE	46 F	4F O	57	36 6
_7 67 g	70 p	78 x	DF	47 G	50 P	58	37 7
_8 68 h	71 q	79 y	E0	48 H	51 Q	59	38 8
_9 69 i	72 r	7A z	E1	49 I	52 R	5A	39 9
_A C4	5E ^ (8)	D2	E2	E8	EE	F4	FA
_B C5	CC	D3	E3	E9	EF	F5	FB
_C C6	CD	D4	E4	EA	FO	F6	FC
_D C7	CE	5B [5D]	EB	F1	F7	FD
_E C8	CF	D6	E6	EC	F2	F8	FE
_F C9	DO	D7	E7	ED	F3	F9	FF

Figure 4-9 (Part 2 of 2). EBCDIC Character Set

	EBCDIC	ASCII
(1)	13 tm	13 dc3
(2)	1C ifs	1C fs
(3)	1D igs	1D gs
(4)	1E irs	1E rs
(5)	1F ius	1F us
(6)	4F	7C
(7)	5F □	7E ~
(8)	9A	5E ^

File

/usr/pub/ebcdic

Related Information

The **dd** command in *AIX Operating System Commands Reference*.

environment

environment

Purpose

Describes the user environment.

Synopsis

Basic Environment

HOME = *path name of home directory*

PATH = *directory search sequence*

TERM = *terminal type*

TZ = *time zone information*

International Character Support Environment

NLFILE = *path name of environment file*

NLCTAB = *path name of collating tables*

NLLANG = *language name*

Japanese Language Support Information

LANG = *current language*

NLSPATH = *path name of a message catalog*

NOSTR = *allowed forms for negative responses*

YESSTR = *allowed forms for affirmative responses*

NLCURSYM = *currency symbol*

NLNUMSEP = *triad and decimal separators*

NLLDAY = *long day names*

NLLMONTH = *long month names*

NLSDAY = *short day names*

NLSMONTH = *short month names*

NLTMISC = *miscellaneous time strings*

NLTSTRS = *relative time names*

NLTUNITS = *time unit names*

NLDATE = *short date format*
NLLDATE = *long date format*
NLTIME = *time format*

Description

When a new process begins, the **exec** system call makes an array of strings available that have the form *name=value*. This array of strings is called the **environment**. Each *name* defined by one of the strings is called an **environment variable** or **shell variable**.

When using the **sh** command interpreter, additional names can be placed in the environment with the **export** or **env** command, or by adding a *name=value* prefix to any other command. See the **sh** command in *AIX Operating System Commands Reference* for more information about setting environment variables with shell commands.

Within a program, the **getenv** subroutine can be used to search the environment for the value of a given variable. The **exec** system call allows the entire environment to be set at one time, usually for a newly started child process.

When creating new environment variables, ensure that their names do not conflict with those of standard variables used by the shell and other programs, such as **MAIL**, **PS1**, **PS2**, and **IFS**.

The Basic Environment

When you log in, a number of environment variables are automatically set by the system before running your login profile, **.profile**. These variables make up the **basic environment**:

- | | |
|-------------|---|
| HOME | The full path name of the user's login or home directory. The login program sets this to the name specified in the /etc/passwd file. |
| PATH | The sequence of directories that commands such as sh , time , nice , and nohup search when looking for a command whose path name is incomplete. The directory names are separated by colons. PATH is set by the system login profile, /etc/profile . |
| TERM | The type of terminal for which output is to be prepared. Commands such as mm and tplot use this information to manipulate special capabilities, if any, of that terminal. The curses , extended curses , and terminfo subroutines also use the value of TERM . For asynchronous terminals, TERM is set by the getty command to a value defined in /etc/ports . For the IBM RT console, TERM is set using the termdef subroutine. |

- TZ** Time zone information. **TZ** is set in the system login profile, */etc/profile*.
- The fields of **TZ** are separated by colons. The first field has three subfields, *lcl*, *n*, and *dst*. If they are not supplied, the defaults for the U.S.A. Eastern Standard (and Daylight Savings) Time Zone are used.
- The additional fields of **TZ** specify when daylight-savings time begins and ends. If these fields are not supplied, U.S.A. rules for daylight-savings time apply. Daylight-savings time specifications apply to all years, and can require adjustment from year to year in locations where the start of daylight-savings time varies.
- TZ** is represented in the format:
- lclndst[:bgn:end:chgwd:chghr:chgamt]*
- where
- | | |
|--------------|---|
| <i>lcl</i> | Is the standard local time zone abbreviation. This name must be 9 bytes or fewer, and cannot contain periods, colons, or hyphens. To be compatible with other operating systems, this name should be 3 bytes. |
| <i>n</i> | Is the difference in local time from GMT in hours (a number from -12 to 12), optionally followed by a period and a number of minutes. Negative differences are for locations east of Greenwich. To be compatible with other operating systems, this difference should not contain minutes. |
| <i>dst</i> | Is the abbreviation for the local daylight-savings time zone, if any. This name must be 9 bytes or fewer, and cannot contain periods, colons, or hyphens. To be compatible with other operating systems, this name should be 3 bytes long. |
| <i>bgn</i> | Is the beginning day of daylight-savings time, if any. This number is the Julian value, or number of days into the year. The value is specified for a non-leap year and adjusted as necessary for leap years. |
| <i>end</i> | Is the ending day (Julian) of daylight-savings time, if any. If the value of <i>end</i> is smaller than the value of <i>bgn</i> , daylight-savings time crosses the new year, as is the practice in the Southern Hemisphere. |
| <i>chgwd</i> | Is the weekday of the change to daylight-savings time, if any. Values range from 1 to 7, with 1 representing Monday and 7 representing Sunday. This value specifies that daylight-savings time begins and ends on the first named weekday <i>before</i> the Julian dates specified by <i>bgn</i> and <i>end</i> . (For example, you could specify the last Saturday in October.) If the value of this field is 0 or not entered, the exact Julian date given is used (corrected for leap year where necessary). |

- chghr* Is the hour of the change to daylight-savings time, if any (number of elapsed hours in the day, optionally followed by a period and a number of minutes).
- chgamt* Is the amount of the change to daylight-savings time. This value is specified by an optionally signed number of minutes, optionally followed by a period and a number of minutes. That is, `[-]hh[.mm]`.

For example, a **TZ** string for Lord Howe Island, Australia in 1985-86 might be:

AusLHIst-10.30AusLHIst:300:60::2:0.30

International Character Support Environment

A special set of environment variables defines the international character support configuration. These environment variables locate configuration information and tailor input and output forms of dates, times, and monetary sums according to “national” or local requirements. If an environment variable *value* for international character support contains blanks, the value appears in quotes; blanks cannot separate the equals sign from the variable name or the value.

Environment variables for international character support are specified in the process environment using ordinary shell environment variables, or in the text file whose path name is specified by the shell environment variable **NLFILE**. Values specified in the process environment take precedence over values specified in **NLFILE**. If a given environment variable is not set either in the process environment or in **NLFILE**, or if a specified value is the null string, a default value is used.

The **NLgetenv** subroutine provides a program with a method to retrieve a value associated with an international character support environment variable.

Environment variables that establish the local environment vocabulary specification consist of a sequence of strings separated by colons. The set of conventions being used is identified by the value of **NLLANG**.

Japanese Language Support Information

The **LANG** conventions replace the **NLLANG** conventions when Japanese Language Support is installed on your system.

Each string must be a translation of the U.S. English name or symbol used in the defaults, in exactly the same order.

The **NLDATE**, **NLLDATE**, and **NLTIME** variables are format strings that can be specified as simple format strings or as **NLstrtime** format strings. These strings are arbitrary, but cannot begin with an * (asterisk). When the patterns listed in the following table appear in a simple format string, **NLgetenv** substitutes the appropriate part of the date or time.

environment

Pattern	Meaning	Replacement
DD	Numeric day of the month	%d
MM	Numeric month	%m
YY	Numeric year (two digits)	%y
YYYY	Numeric year (four digits)	%Y
mon	Month specified by NLSMONTH	%h
month	Month specified by NLSMONTH	%lh
hh	Numeric hour	%H
mm	Numeric minutes	%M
ss	Numeric seconds	%S
aa	Numeric AM/PM indicator	%p

Characters that are not part of replacement patterns are not translated. These are some examples of simple format strings:

mon DD, YYYY	hh.mm
MM/DD/YY	hh:mm:ss
DD.MM.YY	hh:mm aa
YYYY-MM-DD	
DD month YY	

Format strings of the style of **NLstrtime** follow the same form as the *format* parameter of **NLstrtime**, except that the string must be preceded by an asterisk and it cannot contain the formats **%D**, **%sD**, **%lD**, **%T**, **%sT**, or **%r**. The asterisk is not translated and does not become part of the result.

The environment variables are described as follows:

NLFILE The path name of a file containing other environment variable definitions for international character support. **NLFILE** cannot be defined within a file that is identified by another **NLFILE** definition. There is no default path name.

NLCTAB The path name of the file containing tables that define the current collating sequence, as produced by the **ctab** command. The default path name is

/etc/nls/ctab/default

NLLANG The environment language label for the set of variables and environment format strings used for language conventions. The default value is:

u.s.english

Note: Not used in Japanese Language Support.

Japanese Language Support Information

LANG	The current language in effect for Japanese Language Support. The default value is english
NLSPATH	The search path used to locate a message catalog. Each element in the path is a template for a file name that can contain one of the following substitutions: %N The value of the file name, which is passed on to a function that accesses a message catalog %L The value of the LANG variable. A leading or trailing colon, or two adjacent colons, represent the name within the current directory. Following is the default value for NLSPATH : /usr/lpp/msg/%L/%N
NOSTR	The allowed forms for negative responses. A leading or trailing colon, or two adjacent colons, indicate a null response. The order in which the possible responses are listed has no significance. Use the subroutine NLyesno to check an input response against a NOSTR , which has the following default value: n:N:no:No
YESSTR	The allowed forms for positive responses. A leading or trailing colon, or two adjacent colons, indicate a null response. The order in which the possible responses are listed has no significance. Use the subroutine NLyesno to check an input response against a YESSTR , which has the following default value: y:Y:yes:Yes
NLCURSYM	The currency symbol name and placement. The default value is: :\$:L:
NLNUMSEP	The numeric triad and decimal separators. The first of the two separators is the triad separator, which is used to separate groups of three digits in decimal values. The default value for NLNUMSEP is: :,:::
NLLDAY	The full (long) names for the days of the week.

	The default value is: Sunday:Monday:Tuesday:Wednesday:Thursday:Friday:Saturday
NLLMONTH	The full (long) names for the months of the year. The default value is: January:February:March:April:May:June:July:\ August:September:October:November:December
NLSDAY	The short names of the days of the week. Names should be the same length, and of five or fewer characters. The default short name string is: Sun:Mon:Tue:Wed:Thu:Fri:Sat
NLSMONTH	The short names of the months of the year. Names should be the same length, and of five or fewer characters. The default value is: Jan:Feb:Mar:Apr:May:Jun:Jul:Aug:Sep:Oct:Nov:Dec
NLTMISC	Miscellaneous strings needed for input and output of date and time specifications. The default miscellaneous string value is: at:each:every:on:through:am:pm
NLTSTRS	The relative or informal names needed for input of date and time specifications to the remind and at commands (see the remind and at commands in <i>AIX Operating System Commands Reference</i>). The default informal time string value is: now:yesterday:tomorrow:noon:midnight:next:weekdays:weekend
NLTUNITS	The singular and plural forms for all names of units of time, used for input of date specifications to the at command. The default string value for units of time is: minute:minutes:hour:hours:day:days:week:weeks:month:months:\ year:years
NLDATE	The environment format string specifying the short form of the date. This format is used by NLstrtime when the format %D is encountered. The default is: MM/DD/YY
NLLDATE	The environment format string specifying the long form of the date. This form is used by NLstrtime when the formats %lD or %sD are encountered. The default long date format string is: moen DD, YYYY
NLTIME	The environment format string specifying the format of the time; used by NLstrtime when the formats %T , %sT , or %r are encountered.

The default time format string is:

hh:mm:ss

Files

/etc/environment	Sets the basic environment for all processes.
/etc/profile	Allows variables to be added to the environment by the shell.
\$HOME/.profile	Sets the environment for a specific user's needs.
/etc/nls/ctab/default	Sets the international character support environment.

Related Information

In this book: “NLyesno” on page 2-536, “exec: execl, execv, execl, execve, execlp, execvp” on page 2-226, “getenv, NLgetenv” on page 2-347, “NLstrtime” on page 2-530, “NLtmtime” on page 2-533, “termdef” on page 2-814, “passwd” on page 3-167, “profile” on page 3-183, and “TERM” on page 4-76.

The **ctab**, **env**, **export**, **login**, and **sh** commands in *AIX Operating System Commands Reference*.

“Overview of International Character Support” in *Managing the AIX Operating System*.

The discussion of Japanese Language Support in *Japanese Language Support User's Guide*.

eqnchar

eqnchar

Purpose

Identifies special character definitions for **eqn** and **neqn** formatters.

Synopsis

eqn /usr/pub/**eqnchar** [files] | **troff** [options]
neqn /usr/pub/**eqnchar** [files] | **nroff** [options]

Description

The **eqnchar** file contains **troff** and **nroff** character definitions used to construct special scientific symbols. These definitions are primarily intended to be used with the **eqn** and **neqn** formatters. The **eqnchar** file contains definitions for the following characters:

ciplus	⊕			square	□
citimes	⊗	langle	⟨	circle	○
wig	~	rangle	⟩	blot	■
-wig	≡	ppd	⊥	bullet	●
>wig	≧	hbar	ℏ	prop	∝
<wig	≦	<->	↔	empty	∅
=wig	≡	<=>	⇔	member	∈
star	*	<	⊢	nomem	∉
bigstar	*	>	⊣	cup	∪
=dot	≡	ang	∠	cap	∩
orsign	∨	rang	∠	incl	⊆
andsign	∧	3dot	⋮	subset	⊂
=del	≠	thf	∴	supset	⊃
oppA	∇	quarter	1/4	!subset	⊄
oppE	∃	3quarter	3/4	!supset	⊅
angstrom	Å	degree	°	scrL	ℓ
==<	≤	==>	≥		

Figure 4-10. The eqnchar Characters

File

/usr/pub/eqnchar

Related Information

The **eqn**, **nroff**, and **troff** commands in *AIX Operating System Commands Reference*.

fcntl.h

fcntl.h

Purpose

Defines file control options.

Synopsis

```
#include <fcntl.h>
```

Description

The **fcntl.h** header file defines the values that can be specified for the *cmd* and *arg* parameters of the **fcntl** system call, and for the *oflag* parameter of the **open** system call.

```
/* Flag values accessible to open and fcntl */
/* The first three can only be set by open */

#define O_RDONLY 0
#define O_WRONLY 1
#define O_RDWR 2
#define O_NDELAY 04 /* Non-blocking I/O */
#define O_APPEND 010 /* (0x08) append (writes guaranteed */
                    /* at the end) */

/* Flag values accessible only to open */
#define O_CREAT 00400 /* (0x0100) open with create (uses third */
                    /* open arg) */
#define O_TRUNC 01000 /* (0x0200) open with truncation */
#define O_EXCL 02000 /* (0x0400) exclusive open */

/* fcntl requests */
#define F_DUPFD 0 /* Duplicate fildes */
#define F_GETFD 1 /* Get fildes flags */
#define F_SETFD 2 /* Set fildes flags */
#define F_GETFL 3 /* Get file flags */
#define F_SETFL 4 /* Set file flags */
#define F_GETLK 5 /* Get file lock */
#define F_SETLK 6 /* Set file lock */
```

```
#define F_SETLK 7      /* Set file lock and wait */

/* file segment locking set data type - information passed to */
/* system by user */

struct flock {
    short  l_type;
    short  l_whence;
    long   l_start;
    long   l_len;      /* len = 0 means until end of file */
    unsigned long l_sysid;
    short  l_pid;
};

/* file segment locking types */
#define F_RDLCK 01     /* Read lock */
#define F_WRLCK 02     /* Write lock */
#define F_UNLCK 03     /* Remove lock(s) */
```

File

/usr/include/fcntl.h

Related Information

In this book: “fcntl” on page 2-280 and “open” on page 2-538.

fullstat.h

Purpose

Defines the data structure returned by the **fullstat** system call.

Synopsis

```
#include <sys/fullstat.h>
```

Description

The **fullstat.h** header file defines the data structure that is returned by the **fullstat** and **ffullstat** system calls. This file also defines the *cmd* arguments that are used by **fullstat** and **ffullstat**.

```
struct fullstat
{
    /* Beginning of stat block replica... */
    dev_t    st_dev;           /* ID of device containing */
                                /* a directory entry for this file */
                                /* File serial + device uniquely */
                                /* identifies the file within the system */
    ino_t    st_ino;          /* File serial number */
    ushort   st_mode;         /* File mode; see #defines below */
    short     st_nlink;       /* Number of links to file */
    ushort   st_uid;          /* User ID of the owner of the file */
    ushort   st_gid;          /* Group ID of the file group */
    dev_t    st_rdev;         /* ID of this device */
                                /* This entry is defined only for */
                                /* character or block special files */
    off_t     st_size;        /* File size in bytes */
    time_t    st_atime;       /* Time of last access */
    time_t    st_mtime;       /* Time of last data modification */
    time_t    st_ctime;       /* Time of last file status change */
                                /* Time measured in seconds since */
                                /* 00:00:00 GMT, Jan. 1, 1970 */
    /* ...End of stat block replica */
    ushort    fst_uid_raw;     /* Untranslated uid of the file */
}
```

```

ushort  fst_gid_raw;      /* Untranslated gid of the file */
vtype   fst_type;        /* Vnode type */
tagtype fst_uid_rev_tag;  /* uid translation tag */
tagtype fst_gid_rev_tag;  /* gid translation tag */
short *  fst_other_gid_list; /* Pointer to first group ID on */
                                /* alternate concurrent group list */
short    fst_other_gid_count; /* Number of group IDs on */
                                /* alternate concurrent group list */

long     fst_vfs;         /* Virtual file system ID */
long     fst_nid;         /* Node id where the file resides */
int      fst_flag;        /* Indicates whether directory or */
                                /* file is a virtual mount point */
long     fst_i_gen;       /* Inode generation number */
long     fst_reserved[8]; /* Reserved */
};

/* Defines for fullstat or ffullstat cmd argument */
#define FL_STAT      0x0    /* Fullstat */
#define FL_STAT_REV  0x1    /* Fullstat with uid/gid */
                                /* reverse mapping */
#define FL_STAT_OTHER 0x2    /* Reverse mapping, "biased" */
                                /* toward another uid/gid */
/* Defines to tell whether a file or directory is mounted upon */
#define FS_VMP      0x1    /* Virtual mount point */

```

File

/usr/include/sys/fullstat.h

Related Information

In this book: “fullstat, ffullstat” on page 2-334 and “types.h” on page 4-78.

greek

greek

Purpose

Maps Greek characters.

Synopsis

```
cat /usr/pub/greek [ | greek -Tterminal ]
```

Description

The `/usr/pub/greek` file shows the mapping from ASCII characters to the “shift-out” graphics in effect between **SO** and **SI** on TELETYPE Model 37 work stations equipped with an extended (128) character set. These codes are the default Greek characters produced by the **nroff** command. Use the **greek** command to translate these characters for display on other work stations. The file contains:

alpha	α	A		beta	β	B		gamma	γ	\
GAMMA	Γ	G		delta	δ	D		DELTA	Δ	W
epsilon	ϵ	S		zeta	ζ	Q		eta	η	N
THETA	Θ	T		theta	θ	O		lambda	λ	L
LAMBDA	Λ	E		mu	μ	M		nu	ν	@
xi	ξ	X		pi	π	J		PI	Π	P
rho	ρ	K		sigma	σ	Y		SIGMA	Σ	R
tau	τ	I		phi	ϕ	U		PHI	Φ	F
psi	ψ	V		PSI	Ψ	H		omega	ω	C
OMEGA	Ω	Z		nabla	∇	[not	\neg	-
partial	∂]		integral	\int	^				

Figure 4-11. Greek Characters

File

/usr/pub/greek

Related Information

The **300**, **4014**, **450**, **greek**, **hp**, **nroff**, **tc**, and **troff** commands in *AIX Operating System Commands Reference*.

math.h

math.h

Purpose

Defines math subroutines and constants.

Synopsis

```
#include <math.h>
```

Description

This header file contains declarations of all the subroutines in the Math Library (**libm.a**) and of various subroutines in the Standard C Library (**libc.a**) that return floating-point values.

It defines the structure and constants for the **matherr** error-handling mechanism used by the math subroutines. (See “matherr” on page 2-438 for details about this mechanism.)

Among other things, **math.h** defines the following constant, which is used as an error-return value:

HUGE The maximum value of a single-precision floating-point number.

If you define the **_C_func** preprocessor variable before including **math.h**, then **math.h** defines macros that make the names of certain math subroutines appear to the compiler as **_C_XXXX**. The following names are redefined to have a **_C_** prefix:

exp	tan
log	asin
log10	acos
sqrt	atan
sin	atan2
cos	

These special names instruct the C compiler to generate code that avoids the overhead of the math library subroutines and issues compatible-mode floating-point calls directly. See “fpfp” on page 2-295 for information about compatible mode.

The following mathematical constants are also defined for your convenience:

M_E	The base of natural logarithms (e)
M_LOG2E	The base-2 logarithm of e ($\log_2 e$)

M-LOG10E	The base-10 logarithm of e ($\log_{10} e$)
M-LN2	The natural logarithm of 2 ($\log_e 2$)
M-LN10	The natural logarithm of 10 ($\log_e 10$)
M-PI	π , the ratio of the circumference of a circle to its diameter
M-PI-2	The value of $\pi \div 2$
M-PI-4	The value of $\pi \div 4$
M-1-PI	The value of $1 \div \pi$
M-2-PI	The value of $2 \div \pi$
M-2-SQRTPI	The value of 2 divided by the positive square root of π
M-SQRT2	The positive square root of 2
M-SQRT1-2	The positive square root of $\frac{1}{2}$.

The **math.h** file contains an **#include** statement that imbeds another header file named **values.h**. This header file defines a number of machine-dependent constants, and it is discussed on page 4-80.

Files

`/usr/include/math.h`
`/usr/include/values.h`

Related Information

In this book: “matherr” on page 2-438 and “values.h” on page 4-80.

mm

mm

Purpose

Provides the **mm** macro package for formatting documents.

Synopsis

```
mm [options] [files]  
nroff -mm [options] [files]  
nroff -cm [options] [files]  
mmt [options] [files]  
troff -mm [options] [files]
```

Description

This package provides a formatting capability for a very wide variety of documents. How a document is typed and edited on the system is independent of whether the document is to be eventually formatted at a terminal or photoset. See the following references for further details.

Files

```
/usr/lib/tmac/tmac.m  
/usr/lib/tmac/sys.name  
/usr/lib/macros/mm[nt]
```

Related Information

The **mm**, **mmt**, **nroff**, and **troff** commands in *AIX Operating System Commands Reference*.

mptx

Purpose

Provides the macro package for formatting a permuted index.

Synopsis

```
nroff -mptx [flag . . . ] [file . . . ]  
troff -mptx [flag . . . ] [file . . . ]
```

Description

This package provides a definition for the **.xx** macro which is used for formatting a permuted index produced by the **ptx** program. This package does not provide any other formatting capabilities such as headers and footers. Use this macro package in conjunction with the **mm** macro package for these or other capabilities. In this case, the **-mptx** flag must follow the **-mm** flag. For example:

```
nroff -mm -mptx file
```

or

```
mm -mptx file
```

Files

```
/usr/lib/tmac/tmac.ptx  
/usr/lib/macros/ptx
```

Related Information

In this book: “mm” on page 4-66.

The **mm**, **nroff**, **ptx**, **troff** commands in *AIX Operating System Commands Reference*.

Purpose

Provides a **troff** macro package for typesetting view graphs and slides.

Synopsis

```
mvt [-a] [-rw1] [flag . . . ] [file . . . ]  
troff [-a] [-rw1] [-rX1] -mv [flag . . . ] [file . . . ]
```

Description

This package makes it easy to typeset view graphs and projection slides in a variety of sizes. A few macros (briefly described in the following) accomplish most of the formatting tasks needed in making transparencies. The facilities of **troff**, **cw**, **eqn**, and **tbl** are available for more difficult tasks.

The output can be previewed on most terminals. To preview on some devices, specify the **-rX1** option (this option is automatically specified by the **mvt** command, when that command is invoked with certain options). To preview output on other terminals, specify the **-a** option. The **-rw1** option suppresses the printing of cross-hairs and crop marks.

The available macros are:

- .A** [*x*] Places text that follows at the first indentation level (left margin). If 1/2 line spacing was on in the preceding text, *x* suppresses it.
- .B** [*m*[*s*]]
Places text that follows at the second indentation level. Text is preceded by a mark. *m* is the mark, the default is a large bullet. *s* is the increment or decrement to the point size of the mark with respect to the prevailing point size. The default is 0. If *s* is 100, it causes the mark to have the same point size as the default mark.
- .BX** *str1* [*str2*] [*f*]
Encloses *str1* in a box and appends *str2* (if any) to it. *str1* is set in the prevailing font unless *f* names a different font.
- .C** [*m* [*s*]]
Same as **.B**, but for the **third** indentation level. The default mark is a dash.
- .CN** [*args*]
Ends a constant-width font display.

- .CW** [*args*]
Begins a constant-width font display at the current indentation level.
- .D** [*m* [*s*]]
Same as **.B**, but for the fourth indentation level. The default mark is a small bullet.
- .DF** *n f* [*n f* . . .]
Defines font positions. This may not appear within a foil's input text (for example, it may only appear after all the input text for a foil, but before the next foil-start macro). *n* is the position of font *f*, up to four *n f* pairs can be specified. The first font named becomes the prevailing font. The initial setting is (**H** is a synonym for **G**):
 .DF 1 H 2 I 3 B 4 S
- .DV** [*a*] [*b*] [*c*] [*d*] [*e*]
Alters the vertical spacing between indentation levels. The *a*, *b*, *c*, and *d* values alter the spacing for **.A**, **.B**, **.C**, and **.D**, respectively. The *e* value is the pre-spacing and post-spacing for constant-width font displays bracketed by the **.CW** and **.CN** macros. Arguments that are not null must have dimensions. Null arguments leave the corresponding spacing unaffected. Initial setting is:
 .DV .5v .5v .5v 0v .5v
- .I** [*in*] [*a* [*x*]]
Changes the current text indent, but does not affect titles. *in* is the indent in inches, unless dimensioned. The default is 0. If *in* is signed, it is an increment or decrement. The presence of *a* invokes the **.A** macro and passes *x*, if any, to it.
- .S** [*p*] [*l*]
Sets the point size and line length. *p* is the point size, the default is previous. If *p* is 100, the point size reverts to the *initial* default for the current foil-start macro. If *p* is signed, it is an increment or decrement. The default is 18 for **.VS**, **.VH**, and **.SH**, and 14 for the other foil-start macros. *l* is the line length in inches unless dimensioned. The default is 4.2 inches for **.Vh**, 3.8 inches for **.Sh**, 5 inches for **.SH**, and 6 inches for the other foil-start macros.
- .Sh** [*n*] [*i*] [*d*]
Same as **.VS**, except that foil size is 5 by 7 inches.
- .SH** [*n*] [*i*] [*d*]
Same as **.VS**, except that foil size is 7 by 9 inches.
- .Sw** [*n*] [*i*] [*d*]
Same as **.VS**, except that foil size is 7 by 5 inches.
- .SW** [*n*] [*i*] [*d*]
Same as **.VS**, except that foils size is 7 by 5.4 inches.
- .T** *string* Prints *string* as an over-size, centered title.

.U *str1* [*str2*]

Underlines *str1* and concatenates *str2* (if any) to it.

.Vh [*n*] [*i*] [*d*]

Same as **.VS**, except that foil size is 5 by 7 inches.

.VH [*n*] [*i*] [*d*]

Same as **.VS**, except that foils size is 7 by 9 inches.

.VS [*n*] [*i*] [*d*]

Foil-start macro; foil size is to be 7 by 7 inches. *n* is the foil number, *i* is the foil identification, *d* is the date. The foil-start macro resets all parameters (indent, point size, and so on) to initial default values, except for the values of *i* and *d* arguments that came from a previous foil-start macro; it also invokes the **.A** macro.

The naming convention for this and the eight other foil-start macros is that the first character of the name (**V** or **S**) distinguishes between view graphs and slides, respectively, while the second character indicates whether the foil is square (**S**), small wide (**w**), small high (**h**), big wide (**W**), or big high (**H**). Slides are thinner than the corresponding view graphs. For slides, the ratio of the longer dimension to the shorter one is larger than for view graphs. As a result, slide foils can be used for view graphs, but not the opposite. Alternately, view graphs can accommodate a bit more text.

.Vw [*n*] [*i*] [*d*]

Same as **.VS**, except that foil size is 7 inches wide by 5 inches high.

.VW [*n.*] [*i*] [*d*]

Same as **.VS**, except that foil size is 7 by 5.4 inches.

.WS [*w*] [*string*]

Reserves *w* amount of white space. *w* must have dimensions. If *string* is present, it prints the following caption in the reserved space:

Paste up *string* here.

The **.S**, **.DF**, **.DV**, **.U** and **.BX** macros do not cause a break. The **.I** macro causes a break only if it is invoked with more than one argument. All the other macros cause a break.

The macro package also recognizes the following uppercase synonyms for the corresponding lowercase **troff** requests:

.AD .BR .CE .HY .NA .NH .NX .SO .SP .TA .TI

The **Tm** string produces the trademark symbol.

The ~ (tilde) character is translated into a blank on output.

The following **troff** symbols are defined:

- *t** The ASCII tab character.
- *E** The ellipsis (. . .). Do not use this symbol within constant-width text.
- *u** The short name of the operating system in small capital letters.
- *(UU** The short name of the operating system with a leading full-cap letter.
- *(UF** The full name of the operating system.
- *(Tm** The trademark symbol.

Note: The VW and SW foils are meant to be 9 inches wide by 7 inches high. However, the typesetter paper is generally only 8 inches wide, so they are printed 7 inches wide by 5.4 inches high. They need to be enlarged by a factor of 9/7 before they can be used as view graphs.

Files

/usr/lib/tmac/tmac.v
/usr/lib/macros/vmca

Related Information

The **cw**, **eqn**, **mmt**, **tbl**, **troff** commands in *AIX Operating System Commands Reference*.

param.h

param.h

Purpose

Describes system parameters.

Synopsis

```
#include <sys/param.h>
```

Description

Parameters vary among systems using the AIX Operating System. For the RT system, these parameters are in this file. The most significant parameters are:

- | | |
|---------------|--|
| BSIZE | Indicates the kernel buffer size. The RT system has a buffer size of 2048 bytes. This determines the size of block clusters on a file system. Since the size of a block is 512 bytes, a cluster is 4 blocks. |
| NOFILE | Indicates the maximum open file allowed per process. This value is 200. |
| NCARGS | Indicates the maximum number of characters, including terminating null characters that may be passed using the exec system call. |

File

/usr/include/sys/param.h

stat.h

Purpose

Defines the data structure returned by the **stat** system call.

Synopsis

```
#include <sys/stat.h>
```

Description

The **stat** and **fstat** system calls obtain information about a named file. These system calls return a data structure defined by this include file. This file also defines encoding of the **st_mode** field. Note that in the following structure the octal value is shown. The hexadecimal equivalent values are also shown in parentheses.

```
struct stat
{
    dev_t    st_dev;        /* ID of device containing */
                          /* a directory entry for this file */
                          /* File serial + device uniquely */
                          /* identifies the file within the system */
    ino_t     st_ino;        /* File serial number */
    ushort    st_mode;      /* File mode; see #defines below */
    short     st_nlink;     /* Number of links to file */
    ushort    st_uid;       /* User ID of the owner of the file */
    ushort    st_gid;       /* Group ID of the file group */
    dev_t     st_rdev;      /* ID of this device */
                          /* This entry is defined only for */
                          /* character or block special files */
    off_t     st_size;      /* File size in bytes */
    time_t    st_atime;     /* Time of the last access */
    time_t    st_mtime;     /* Time of the last data modification */
    time_t    st_ctime;     /* Time measured in seconds since */
                          /* 00:00:00 GMT, Jan. 1, 1970 */
};
```

```
#define S_IFMT    0170000        /* (0xF000) type of file */
#define S_IFDIR   0040000        /* (0x4000) directory */
#define S_ISDIR(m) (((m) & (S_IFMT)) == (S_IFDIR))
#define S_IFCHR   0020000        /* (0x2000) character special */
#define S_ISCHR(m) (((m) & (S_IFMT)) == (S_IFCHR))
#define S_IFBLK   0060000        /* (0x6000) block special */
#define S_ISBLK(m) (((m) & (S_IFMT)) == (S_IFBLK))
#define S_IFREG   0100000        /* (0x8000) regular */
#define S_ISREG(m) (((m) & (S_IFMT)) == (S_IFREG))
#define S_IFIFO   0010000        /* (0x1000) fifo */
#define S_ISFIFO(m) (((m) & (S_IFMT)) == (S_IFIFO))
#define S_ISUID   04000 /* (0x0800) set user id on execution */
#define S_ISGID   02000 /* (0x0400) set group id on execution */
#define S_ISVTX   01000 /* (0x0200) save swapped text even after use */
#define S_IRWXU   00700 /* (0x01C0) owner read,write,execute permission */
#define S_IREAD   00400 /* (0x0100) owner read permission */
#define S_IRUSR   00400 /* (0x0100) read permission, owner */
#define S_IWRITE  00200 /* (0x0080) owner write permission */
#define S_IWUSR   00200 /* (0x0080) owner write permission */
#define S_IXEXEC  00100 /* (0x0040) owner execute/search permission */
#define S_IXUSR   00100 /* (0x0040) owner execute/search permission */
#define S_IRWXG   00070 /* (0x0038) group read,write,execute permission */
#define S_IRGRP   00040 /* (0x0020) group read permission */
#define S_IWGRP   00020 /* (0x0010) group write permission */
#define S_IXGRP   00010 /* (0x0008) group execute/search permission */
#define S_IRWXO   00007 /* (0x0007) other read,write,execute, permission */
#define S_IROTH   00004 /* (0x0004) other read permission */
#define S_IWOTH   00002 /* (0x0002) other write permission */
#define S_IXOTH   00001 /* (0x0001) other execute/search permission */
#define S_IFMPX   S_IFCHR|S_ISVTX /* multiplex character special file */
#define S_ISMPX(m) (((m) & (S_IFMT|S_ISVTX)) == (S_IFMPX))
#define S_ENFMT   S_ISGID /* record locking enforcement flag */
```


Examples

The **S_IREAD**, **S_IWRITE**, and **S_IEXEC** masks can be used to test permissions in any of the three groups (owner, groups, or other) by shifting them. For example, to test for read access by group, use:

```
st_mode & (S_IREAD >> 3)
```

To test for global write access, use:

```
st_mode & (S_IWRITE >> 6)
```

File

`/usr/include/sys/stat.h`

Related Information

In this book: “stat, fstat, lstat” on page 2-786 and “types.h” on page 4-78.

TERM

TERM

Purpose

Lists conventional names for terminals.

Description

These names are used primarily for commands such as **mm** and **nroff**. These names are maintained as part of the shell environment in the variable **TERM**. See the **sh** command in *AIX Operating System Commands Reference* for an explanation of the shell. Also see “profile” on page 3-183 and “environment” on page 4-48 in this book for use of the **TERM** environment variable.

TERM	Terminal Description
ibm3151	IBM 3151 Asynchronous Terminal
ibm3161	IBM 3161 ASCII Display
ibm3161	IBM 3163 ASCII Display
ibm3161-C	IBM 3161 ASCII Display with cartridge (for international character support)
ibm3162	IBM 3162 ASCII Display (for international character support)
ibm5081	IBM 5081 Display
ibm5081-56	IBM 5081 Display using a 56-column font
ibm5081-113	IBM 5081 Display using a 113-column font
ibm5151	IBM 5151 Display and Printer Adapter with IBM 5151 Display
ibm5154	IBM 5154 Display Adapter with IBM 5151 Display
ibm5154	IBM 5154 Display Adapter with IBM 5154 Display
ibm5154	IBM 6154 Display Adapter with IBM 6154 Display
ibm5555 em	IBM 5550 Japanese PC terminal emulator
ibm6153	IBM 6153 Display Adapter with IBM 6153 Display
ibm6153-40	IBM 6153 Display using a 40-column font
ibm6153-90	IBM 6153 Display using a 90-column font
ibm6154	IBM 6154 Display
ibm6154-40	IBM 6154 Display using a 40-column font
ibm6154-90	IBM 6154 Display using a 90-column font
ibm6155	IBM 6155 Display
ibm6155-56	IBM 6155 Display using a 56-column font
ibm6155-113	IBM 6155 Display using a 113-column font
vt100	DEC VT100
vt220	DEC VT220
dumb	Terminal types with no special features (such as reverse line motion) (implies -c)

lp	Line Printer (implies -c) (must pipe through lpr or some such filter)
37	Teletype Model 37 KSR
42	ADM 42 (implies -c)
300	DASI (DTC, GSI) 300
300s	DASI 300s
300-12	DASI 300 at 12-pitch
300s-12	DASI 300s at 12-pitch
tn300	TermiNet 300 (implies -c)
382	DTC 382
450	DASI 450 (same as Diablo 1620) DEFAULT
450-12	DASI 450 (same as Diablo 1620) 12-pitch
2631	HP 2631 series line printer (implies -c)
2631-e	HP 2631 series (expanded mode) (implies -c)
2631-c	HP 2631 series (compressed mode) (implies -c)
4000a	Trendata 4000a

Up to eight characters chosen from `[-a-z0-9]` make up a basic terminal name. Terminal submodels and operational modes are distinguished by suffixes beginning with a `-` (hyphen). Names should generally be based on original vendors, rather than local distributors. A terminal acquired from one vendor should not have more than one distinct basic name.

Commands whose behavior depends on the type of terminal should accept parameters such as **-Tterm** where *term* is one of the names in the preceding list. If the parameter is not in the list, the commands should obtain the terminal type from the environment variable **TERM**, which in turn should contain *term*. Any unknown terminal is treated as a **dumb** terminal.

This list does not include all supported terminals. See “terminfo” on page 3-219 for additional **TERM** variables.

File

`/user/lib/help/term`

Related Information

In this book: “environment” on page 4-48 and “terminfo” on page 3-219.

The **mm**, **sh**, **stty**, **tabs**, **nroff**, and **environ** commands in *AIX Operating System Commands Reference*.

types.h

types.h

Purpose

Defines data types for the system.

Synopsis

```
#include <sys/types.h>
```

Description

The data types defined in this include file are used in the RT system source code. Some data of these types are accessible to user code:

```
typedef struct {int r[1];} * physadr;
typedef long                level_t
typedef long                daddr_t;
typedef char *              caddr_t;
typedef unsigned int        uint;
typedef unsigned short      ushort;
typedef unsigned long       ulong;
typedef ushort              ino_t;
typedef short               cnt_t;
typedef long                time_t;
typedef int                 label_t[11];
typedef int                 dev_t;
typedef long                off_t;
typedef long                paddr_t;
typedef long                key_t;
```

Notes:

- daddr_t** This data type is used for disk addresses, except in i-nodes on disk. See the "fs" on page 3-100 for the format of disk addresses used in i-nodes.
- time_t** Times are encoded in seconds since 00:00:00 GMT, January 1, 1970.
- dev_t** The major and minor parts of a device code specify kind of device and unit number of the device, and they depend on the system customization.

off_t Offsets are measured in bytes from the beginning of a file.
label_t Variables of this type are used to save the processor state while another process is running.

File

`/usr/include/sys/types.h`

Related Information

In this book: “fs” on page 3-100 and “values.h” on page 4-80.

Purpose

Defines machine-dependent values.

Synopsis

```
#include <values.h>
```

Description

This header file contains a set of manifest constants that are conditionally defined for particular processor architectures. The model for integers is assumed to be a one's- or two's-complement binary representation, in which the sign is represented by the value of the high-order bit.

BITS (<i>type</i>)	The number of bits in the specified data type
HIBITS	A short integer with only the high-order bit set (0x8000)
HIBITL	A long integer with only the high-order bit set (0x80000000)
HIBITI	A regular integer with only the high-order bit set (the same as HIBITL)
MAXSHORT	The maximum value of a signed short integer (0x7FFF \equiv 32767)
MAXLONG	The maximum value of a signed long integer (0x7FFFFFFF \equiv 2147483647)
MAXINT	The maximum value of a signed regular integer (the same as MAXLONG)
MAXFLOAT	The maximum value of a single-precision floating-point number
MAXDOUBLE	The maximum value of a double-precision floating-point number
LN_MAXDOUBLE	The natural logarithm of MAXDOUBLE
MINFLOAT	The minimum positive value of a single-precision floating-point number
MINDOUBLE	The minimum positive value of a double-precision floating-point number

FSIGNIF	The number of significant bits in the mantissa of a single-precision floating-point number
DSIGNIF	The number of significant bits in the mantissa of a double-precision floating-point number
FMAXEXP	The maximum exponent of a single-precision floating-point number
DMAXEXP	The maximum exponent of a double-precision floating-point number
FMINEXP	The minimum exponent of a single-precision floating-point number
DMINEXP	The minimum exponent of a double-precision floating-point number
FMAXPOWTWO	The largest power of two that can be exactly represented as a single-precision floating-point number
DMAXPOWTWO	The largest power of two that can be exactly represented as a double-precision floating-point number.

File

`/usr/include/values.h`

Related Information

In this book: “math.h” on page 4-64, “types.h” on page 4-78.

Chapter 5. Special Files

About This Chapter

This chapter describes various special files that refer to specific hardware peripherals and RT system device drivers. The names of the entries are generally derived from names for the hardware as opposed to the names of the special files themselves. Characteristics of both the hardware device and the corresponding RT system device driver are discussed where applicable.

asy

Purpose

Supports the asynchronous adapter.

Description

The **asy** driver supports asynchronous ports. If a port is not installed, an attempt to open it fails. Each port can be individually programmed for speed (50-19.2K baud), character length, and parity. Output speed is always the same as input speed. The behavior of each adapter is described in the **termio** file.

The asynchronous port is a character-at-a-time device for both input and output. This characteristic limits the bandwidth, which can be achieved over a line and increases the interrupt loading on the central processor.

If the port was opened with the modem control bit present in the minor device (see the following text), modem control is enabled. If enabled, the driver waits in the **open** routine until data carrier detect is present. Once opened, if data carrier detect drops, the driver returns errors on any subsequent user read or write attempts of the asynchronous port. If the port was opened as a controlling teletype, a **SIGHUP** signal is generated to the process that opened the port.

Minor Device Numbers

The asynchronous ports are character devices. The low-order bit of the minor device number corresponds to the primary or secondary asynchronous ports. Bit 6 enables modem control on the selected port. Thus, minor device 0 corresponds to the first asynchronous port with modem control disabled, while minor device 65 corresponds to the second asynchronous port with modem control enabled.

Files

/dev/tty*	for remote devices.
/dev/ltty*	for local devices.

Related Information

In this book: “termio” on page 5-133 and “signal” on page 2-750.

The **config** command in *AIX Operating System Commands Reference*.

audit

Synopsis

```
#include <sys/audit.h>
```

Purpose

Provides stream interface to audit records.

Synopsis

Description

The **audit** file is a multiplexed device interface to the audit system. Each **open** creates a new channel on which audit records can be read. Each channel specifies the audit classes for which audit records are to appear on the device. These classes must have been defined by the **auditevents** system call. System auditing need not be active when the channel is created.

Each **read** system call used on this file will be blocked until auditing is enabled and data is available. If no data is buffered on the channel and auditing is disabled, the **read** operation returns an end-of-the-file (EOF) indicator.

By default, all audit events appear in the stream. You can restrict the list of classes by using the *ext* parameter on the **openx** system call, or the **AIO&US.EVENTS** subcommand of the **ioctl** system call.

```
openx("/dev/audit", "0,  
ext)
```

Where *ext* is a pointer to a structure:

```
{  
    char *events;  
}
```

The *events* parameter is a list of **NULL**-separated audit class names.

audit

Note that specifying events for the stream does not cause auditing of those events to be enabled.

There are three minor devices for the audit device:

Minor 0 The driver will drop a record when there is no room in the buffer to hold it.

Minor 1 The driver will block the process if there is not enough room in the buffer to hold the record. The process will resume when there is sufficient space to write the record.

Minor 2 The driver will write the record then block the process until the record is read. This minor device is used for threat monitoring.

ioctl Operations

The **AIO_EVENTS** type of **ioctl** operation specifies the audit classes to be recorded on a channel of the audit device. This operation has the form:

```
ioctl(fd,AIO_EVENTS,events)
char *events;
```

The *events* parameter is a list of **NULL**-separated audit class names.

The **IOCTYPE** type **ioctl** system call returns the value **DD_AUDIT**, which is defined in **sys/devinfo.h**.

The **IOCINFO** type **ioctl** system call initializes only the **devtype** field of the **devinfo** structure, which is defined in **sys/devinfo.h**.

File

/dev/audit

Related Information

The following system calls: “audit” on page 2-31 and “auditevents” on page 2-36 The **watch** command.

biodd

Purpose

Provides a programming interface for transmitting and receiving data for adapters that are accessed through the VRM Block I/O Device Manager.

Synopsis

```
#include <sys/biodd.h>
```

Description

The Block I/O Kernel Device Driver provides support for transmitting and receiving data for adapters that are accessed through the VRM Block I/O Device Manager. This driver is designed to be a multiplexed device driver that permits multiple opens for those adapters allowed to have multiple opens by VRM device driver.

The Block I/O Kernel Device Driver can be used by any application that is written to run above the kernel. The **/etc/biodd** file contains the device names of the adapters to be accessed and can be edited to add or remove device names. After **/etc/biodd** has been edited, the device driver must be configured for the adapters that the user wants to access by means of the Block I/O Kernel Device Driver using the **biodd_cfg** command. To configure these adapters automatically each time the system is started, uncomment the **/etc/biodd_cfg** line in the **/etc/rc.include** file.

A process interfaces to the Block I/O Kernel Device Driver through the following system calls:

- | | |
|---------------|---|
| open | Initializes the path to an adapter port for the kernel process. |
| close | Indicates the port will no longer be used and deletes the path from the kernel process. |
| read | Receives data from the adapter. |
| write | Sends data to the adapter. |
| select | Waits on file descriptors or message queue IDs until one is ready. |
| ioctl | Used for the device-dependent commands described in "ioctl Operations" on page 5-12. |

Error Handling

There are two types of errors that the Block I/O Kernel Device Driver is unable to associate with a specific command: write command errors and unsolicited interrupts that are not related to a specific command. Write command error information may be received after return has already been made to the user. This condition occurs because, for the block I/O **write** system call, return simply means that the command has been enqueued to the VRM and not that the command has actually been executed by the VRM. Also, certain unsolicited interrupts occur that are not related to a specific command. When the Block I/O Kernel Device Driver receives one of these interrupts, the completion information is saved and a return code of -1 and an **errno** value of **EIO** are returned to the user on the next I/O request for that particular file descriptor. The I/O request will not be acted upon and must be reissued by the user.

Completion information returned to the user by the **BIOC_GET_PSB ioctl** call indicates the error type. For more information on the data returned from the program status block (PSB), see *VRM Device Support*. The PSB status flags field contains bits that indicate this information pertains to a certain interrupt type. If the interrupt type is a **Send Command I/O** interrupt, the return information includes an operation options field that can be used to identify the specific command. See *VRM Device Support* for an explanation of the PSB status flags and the operation options.

To receive notification at the time an error occurs, rather than waiting until the next I/O request, use the **select** system call to select exceptions for this file descriptor. You can also use the **BIOC_SET_SIGNAL ioctl** system call to set a signal value that is sent whenever an exception condition occurs. The signal can then be handled in the normal manner.

Once you are notified of an error, you are not notified again of the same error. For example, if you are notified of an unsolicited interrupt by means of a signal or in response to a **select** system call, you do not receive an indication of that error on the next I/O request.

open Support

The **open** system call opens a file descriptor for the file named by the *path* parameter. The *path* specifies the Block I/O Kernel Device Driver (**/dev/biodd**) followed by a / (slash) and the device name (from the **devices** command).

The **open** system call has the following format:

```
int open (path, oflag)  
char *path;  
int oflag;
```

The value of *oflag* sets the file status flag. This value is constructed by logically ORing flags from the following list:

- O_RDWR** Open for reading and writing.
- O_RDONLY** Open for reading only.
- O_WRONLY** Open for writing only.
- O_NDELAY** If set, a **read** system call returns immediately either with or without data. If this flag is not set, a **read** system call waits if there is no data to return.

To complete the **open** process, the **open** system call must be followed by a **BIOC-START ioctl**.

Upon successful completion, a file descriptor is returned. If the **open** fails, a value of -1 is returned and **errno** is set to indicate the error.

The **open** fails if one or more of the following are true:

- ENOENT** The named file does not exist.
- ENOTDIR** A component of the *path* prefix is not a directory.
- EACCES** A component of the *path* prefix denies search permission, or permission is denied for the named file.
- EISDIR** The named file is a directory and *oflag* is write or read/write.
- EROFS** The named file resides on a read-only file system and *oflag* is write or read/write.
- EMFILE** Maximum number of file descriptors are currently open.
- EFAULT** The *path* parameter points outside the process's allocated address space.
- EINVAL** Invalid parameter or stanza name too long (greater than 32) or not supported.

Note: For an adapter card that allows only one **open** per port, this error is returned if the port is already open.
- EINTR** The **open** system call was interrupted.
- ENODEV** There is no such device.
- EBUSY** All of the available 256 ports are in use.
- EPERM** The **open** for device **CONFIG** requires superuser authority.
- ENOMEN** The device driver was unable to allocate space it needed.
- ENXIO** An attempt was made to **open** the device driver before it was configured.

read Support

The **read** system call is used to receive data from an application. A pointer of *buf* points to a buffer area into which the data is read. This buffer address must be on a word boundary. The receive data area is preceded by an 80-byte header containing the data length and offset fields of the receive data. For a complete description of the data buffer structure for a specific adapter, see *VRM Device Support*.

The value of *nbyte* specifies the maximum amount of data the program receives. The value of *nbyte* should be greater than or equal to the device buffer length. The device buffer length is set by running the **devices** command for the specified adapter and setting the **lobibp** parameter. If the data to be received is larger than *nbyte*, an error of **EINVAL** is returned.

The **read** system call has the following format:

```
int read (fildes, buf, nbyte)
int fildes;
char *buf;
int nbyte;
```

Upon successful completion, a positive integer is returned indicating the number of bytes read. An unsuccessful **read** returns a value of -1 and **errno** is set to indicate the error.

The **read** fails if one or more of the following are true:

- | | |
|---------------|---|
| EFAULT | An invalid address was specified. |
| EIO | An I/O error occurred. This error was outstanding at the time the read was issued. A BIOC_GET_PSB ioctl system call will supply more information. |
| EBADF | An invalid file descriptor was specified. |
| EINTR | The read system call was interrupted. |
| EINVAL | An invalid argument was specified. |
| ENODEV | An attempt to read was made before the device was started. |

write Support

The **write** system call sends data to an application. The **Write-Send-Command** SVC is used by the Block I/O kernel device driver to send the data to the VRM.

A pointer of *buf* points to the buffer area to send. This buffer address must be on a word boundary. An 80-byte buffer header must precede the data to be written. The data length and data offset fields and, if present, device-dependent information for the adapter, must be

set by the user. For a complete description of the data buffer structure of a specific adapter, see *VRM Device Support*.

The value of *nbyte* specifies the buffer length, in bytes, that the program is sending. The value of *nbyte* must be less than the device buffer length. The device buffer length is set with the **lobibp** parameter of the **devices** command. If *nbyte* is larger than the device buffer length, an error of **EINVAL** is returned.

The **write** system call has the following format:

```
int write (fildes, buf, nbyte)
int fildes;
char *buf;
int nbyte;
```

Upon successful completion, a positive integer is returned indicating the number of bytes written. An unsuccessful **write** returns a value of -1 and **errno** is set to indicate the error.

The **write** fails if one or more of the following are true:

- EBADF** An invalid file descriptor was specified.
- EFAULT** An invalid address was specified.
- EIO** An I/O error occurred. This error was outstanding at the time the **write** was issued. A **BIOC_GET_PSB ioctl** system call will supply more information.
- EINVAL** An invalid parameter was passed.
- EINTR** The **write** system call was interrupted.
- ENODEV** An attempt to **write** was made before the device was started.
- ENOSPC** There is currently no space available in the VRM buffer pool. Data cannot be enqueued to the VRM at this time.

Note: The total number of buffers (in the VRM buffer pool and available to the device) is set with the **nobibp** parameter of the **devices** command.

close and select Support

For a complete description of the **close** and **select** system calls, see “close” on page 2-101 and “select” on page 2-703.

ioctl Operations

The Block I/O Kernel Device Driver performs the following **ioctl** operations. For a complete description of the **ioctl** system call, see “**ioctl**” on page 2-407

Four Block I/O Kernel Device Driver **ioctl** calls have the format:

```
int ioctl (fildes, command, arg)
int fildes;
int command;
char *arg;
```

The four commands using this format are:

BIOC_GET_PSB Returns completion information from the Program Status Block about an exception condition that has occurred. The program is notified that the information is available either by receiving a negative return code and an **errno** value of **EIO** from a Block I/O Kernel Device Driver system call, or by a return from a **select** system call that is waiting due to exceptions.

The *arg* parameter contains the address of the buffer into which the completion information returns. This buffer must be at least 20 bytes in length and, on return, contains the following general information:

Reserved	2 bytes
Status flags	1 byte
Overrun count	1 byte
Status word	Consists of the following for this application: Operation results (2 bytes) Input/Output device number or <i>iodn</i> (2 bytes)
Data word 1	4 bytes
Data word 2	4 bytes
Data word 3	4 bytes

For a description of the specific SVC command completion information contained in the above fields, see *VRM Device Support*.

Upon successful completion, a value of 0 is returned. If the **BIOC_GET_PSB** **ioctl** fails, a value of -1 is returned and **errno** is set to indicate the error.

The **BIOC_GET_PSB** **ioctl** fails if one or more of the following are true:

EBADF	An invalid file descriptor was specified.
EFAULT	An invalid address was specified.
EINVAL	There is no error PSB available to send back.

BIOC_QUERYDEV Returns the Query Device Structure (QDS) for the stated device. This structure is returned from a **Query_Device** SVC to the VRM. The *arg* parameter contains the address of a user input structure containing the following fields:

Reserved	This field is 2 bytes long.
Query options	Defines the device information to return. This is a 2-byte field. The possible values are: <ul style="list-style-type: none"> 1 Returns the hardware characteristics 2 Returns the device characteristics 4 Returns the error log 7 Returns all of the above.
QDS length	Contains the length, in bytes, of the buffer in which the QDS returns. If the buffer is not large enough to contain the requested information, an error returns. This is a 4-byte field.
QDS address	Contains the address of the buffer in which the requested QDS information returns. This is a 4-byte field. The QDS returned consists of Define Device Structure (DDS) header information, operation completion information for the last operation completed, and the device-dependent information requested by the query options field. For more information about the device-dependent information, see <i>VRM Device Support</i> .

If an error occurs, a value of -1 is returned and **errno** is set to indicate the error. A value of 0 is returned if there is no error.

The **BIOC_QUERYDEV ioctl** fails if one or more of the following are true:

EBADF	An invalid file descriptor was specified.
EINTR	The ioctl system call was interrupted.
EINVAL	The length of the Query Device Structure provided is not large enough to contain the requested data.
EFAULT	An invalid address was specified.
ENOMEM	The device driver was unable to allocate needed space.
EIO	An I/O error occurred. This error was outstanding at the time the query was issued. A BIOC_GET_PSB ioctl system call will supply more information.

BIOC_SEND_COMMAND

Issues a device-dependent send command. This request is done by requiring the user to provide all the needed information to the **Send_Command** SVC to the VRM.

The *arg* parameter contains the address of a 24-byte (6-word) buffer. This buffer is used both for input data and to return data to the user. On input, the 6-word buffer should contain the data passed in General Purpose Registers 2 through 7 (GPR2-GPR7). For more information on device-dependent commands and the input and return data for a specific adapter, see *VRM Device Support*.

The contents of the input structure are described as follows:

GPR2 Bits 0-15 of the first word are set equal to the input/output device number (IODN) of the device by the Block I/O Kernel Device Driver. Any value put in this area by the user will be overwritten.

Bits 16-31 contain the operation options field. With the exception of bits 19 and 24-31, this field is set by the kernel device driver. Bits 19 and 24-31 are set by the user as follows:

Bit 19 Sets the command extension.

Bits 24-31 Sets the device option for the device-dependent command to a value greater than 8. The kernel device driver checks that this field is greater than 8 in order to ensure that this is a device-dependent command.

GPR3 The second word must be set equal to the command-specific data for the device.

GPR4 The third word must be set equal to the command-specific data for the device.

GPR5 The fourth word must be set equal to the address of the command extension area, if one is required for this device-dependent command; otherwise, this word contains command-specific data. If a command extension area is required for this command, the user must also fill in the necessary fields of this buffer.

GPR6 The fifth word must be set equal to the length of the command extension area, if one is required for this device-dependent command; otherwise, this word contains command-specific data.

Note: Some device-dependent commands can also return data in the command extension area whose address was passed on the call. When this occurs, the command extension area must be long enough to include all returned data.

GPR7 The sixth word will be set equal to the path identifier of the path to the VRM device driver by the Block I/O Kernel Device Driver.

If the command completes successfully, upon return, the buffer pointed to by the *arg* parameter contains Program Status Block completion information. For information on this completion data, see *VRM Device Support*.

If an error occurs, a value of -1 is returned and **errno** is set to indicate the error. A successful completion returns a value of 0.

The **BIOC_SEND ioctl** does not complete successfully if one or more of the following are true:

EBADF An invalid file descriptor was specified.

EINVAL An invalid parameter was passed. This command is not one of the device-dependent commands. (The device option specified is less than or equal to 8.)

EFAULT An invalid address was specified.

EINTR The **ioctl** system call was interrupted.

EIO An I/O error occurred. This error can be an error obtained on the execution of this system call, or it can be an error that was outstanding at the time this call was issued. A **BIOC_GET_PSB ioctl** system call will supply more information.

ENOMEM The device driver was unable to allocate needed space.

BIOC_START

Requests the VRM device driver to start a specified network ID. For the Token-Ring adapter, this is the service access point (SAP) and for the baseband adapter, this is the type field. Only one **BIOC_START ioctl** system call is allowed per open.

The *arg* parameter contains the address of a 24-byte (6-word) buffer. This buffer is used both for input data and to return data to the user. On input, the 6-word buffer should contain the data passed in General Purpose Registers 2 through 7 (GPR2-GPR7). For more information on the **Start Send SVC** command, see *VRM Programming Support* and for more information on the input and return data for a specific adapter, see *VRM Device Support*.

The contents of the structure are described as follows:

GPR2 Bits 0-15 of the first word will be set equal to the input/output device number (IODN) of the device by the Block I/O Kernel Device Driver. Any value put in this area by the user is overwritten.

Bits 16-31 contain the operation options field. This field is set by the kernel device driver.

- GPR3** Reserved for system use.
- GPR4** Reserved for system use.
- GPR5** The fourth word must be set equal to the address of the command extension area.
- GPR6** The fifth word must be set equal to the length of the command extension area.

Note: Some **Start Send** SVC commands can also return data in the command extension area whose address was passed on the call. When this occurs, the command extension area must be long enough to include all returned data.

- GPR7** The sixth word is set equal to the path identifier of the path to the VRM device driver by the Block I/O Kernel Device Driver.

The following list describes the fields within the command extension area that are set by the kernel device driver or the user:

- Level/sublevel** The interrupt level used to return virtual interrupts to the kernel device driver. (Set by the kernel device driver)
- Ring correlator** A 1-byte value returned by the VRM with all virtual interrupts for the following network ID. (Set by the kernel device driver)
- device dependent** An optional area used to pass additional device-dependent information. For a description of the device-dependent area of the command extension area, see the **Start_Device** SVC command for the specific VRM device driver in *VRM Device Support*.

If the command completes successfully, upon return, the buffer pointed to by the *arg* parameter contains Program Status Block completion information. For information on this completion data, see *VRM Device Support*.

If an error occurs, a value of -1 is returned and **errno** is set to indicate the error. A successful start returns a 0. The completion information returned to the user is only valid if the user receives a return code of 0.

The **BIOC_START ioctl** does not complete successfully if one or more of the following are true:

EBADF	An invalid file descriptor was specified.
EINVAL	An invalid parameter was passed.
EFAULT	An invalid address was specified.
EINTR	The ioctl system call was interrupted.
EBUSY	A START has already been performed for this file descriptor and only one start command is allowed per file descriptor.
EIO	An I/O error occurred. This error may be an error obtained on the execution of this system call, or it may be an error that was outstanding at the time this call was issued. A BIOC_GET_PSB ioctl system call will supply more information.
ENOMEM	The device driver was unable to allocate needed space.

The command, **BIOC_SET_SIGNAL**, has the **ioctl** system call format:

```
int ioctl (fildes, BIOC_SET_SIGNAL, arg)
int fildes;
int BIOC_SET_SIGNAL;
int arg;
```

The command, **BIOC_SET_SIGNAL** sets a signal value of *arg* that is to be sent whenever an exception condition is encountered. A value of **NO_SIG** turns off signals. For descriptions of signal values, see the **/usr/include/sys/signal.h** file.

The command **IOCCONFIG** uses the **ioctl** system call format:

```
#include <sys/ioctl.h>

int ioctl (fildes, IOCCONFIG, arg)
int fildes;
int IOCCONFIG;
struct config *arg;
```

The command, **IOCCONFIG**, configures the device driver for all devices that appear in the **/etc/biodd** file. An AIX command that issues this call is installed with the device driver. The command, **/etc/biodd-cfg**, is usually called from the **etc/rc.include** file which runs at IPL.

biodd

This command copies the internal table, which ties the stanza to the device IODN, into the device driver. A pointer of *arg*, to a **config** structure, contains the number of devices and the stanza name and IODN for each device.

The command **IOCINFO** has the **ioctl** system call format:

```
int ioctl (fildes, IOCINFO, arg)
int fildes;
int IOCINFO;
struct devinfo *arg;
```

The command, **IOCINFO**, returns a **devinfo** structure that describes the device type and its associated flags. The file descriptor returned from an open is specified by *fildes*. For the format of this structure, see **/usr/include/sys/devinfo.h**.

If an error occurs, a value of -1 is returned and **errno** is set to indicate the error. Otherwise, a **devinfo** structure with the device type and flags is returned. The value of *flags* will be zero.

The command, **IOCTYPE**, has the **ioctl** system call format:

```
int ioctl (fildes, IOCTYPE)
int fildes;
int IOCTYPE;
```

The command **IOCTYPE**, returns the device type associated with the file descriptor *fildes*. If an error occurs, a value of -1 is returned and **errno** is set to indicate the error. If no errors occur, the device driver type returns.

Files

/dev/biodd Device driver file.
/etc/biodd Configuration file.

Related Information

In this book: “close” on page 2-101, “ioctl” on page 2-407, “open” on page 2-538, “read, readx” on page 2-591, “select” on page 2-703, “write, writex” on page 2-876, and “system” on page 3-210.

VRM Programming Support

VRM Device Support

The **biodd_cfg** command in *AIX Operating System Communications Guide*.

bus

Purpose

Supports the hardware bus interface.

Synopsis

```
#include <sys/hwdbus.h>
#include <fcntl.h>
```

Description

The **bus** file consists of a pseudo-driver in the kernel, which allows a user program to enable the hardware I/O bus such that the address space can be addressed directly by the program rather than performing I/O through the AIX system calls.

The user program first opens the special file name associated with the device driver. Only **O_RDONLY**, **O_WRONLY**, and **O_RDWR** are valid values to be used with the **open** system call. Before the open, any addressing of I/O space generates a SIGSEGV signal.

Once the device driver is open, the user program should issue an **ioctl** system call to obtain the base addresses of the bus I/O and bus memory spaces. These base addresses are added to the appropriate address offsets to obtain an absolute I/O address.

ioctl Operations

The **ioctl** system call is invoked as follows:

```
ioctl (fildes, command, ptr) /* get addresses */
int fildes;                  /* file descriptor */
int command;                 /* valid value is HWDBASE */
struct hwdbase               /* contains base addresses */
{
    char *hwdio;
    char *hwdmem;
} *ptr
```

The **hwdio** field is the address of the start of I/O memory allocated to the I/O port address. The **hwdmem** field is the address of the start of I/O memory allocated to dedicated memory, such as display refresh memory.

bus

File

`/dev/bus`

Related Information

In this book: “hft” on page 5-39.

config

Purpose

Configures system device drivers.

Synopsis

```
#include <sys/types.h>
#include <sys/kcfg.h>
#include <sys/ksvc.h>
```

Description

The **config** driver is used to customize the VRM and the kernel. Use of this pseudo-device is restricted to a user with superuser authority. Most operations are performed via **ioctl** system calls. The **write** system calls are accepted only in certain situations as described.

ioctl Operations

A list of **ioctl** calls along with the descriptions follows. These calls return a value of 0 upon successful completion. Otherwise, a value of -1 is returned and **errno** is set to indicate the error.

If an error occurred while issuing a supervisor call (SVC) to the VRM, **errno** is set to **EIO**. The VRM status code is obtained using the **CFRSTAT** **ioctl** call. In all the following calls that pass a structure address, the call is aborted if the structure is not entirely within memory that is addressable by the calling process. In that case, **errno** is set to **EFAULT**.

CFBUFF Allocates and initializes the Block I/O Communication Area (BIOCA) in kernel memory. The *arg* parameter is a pointer to a **defdev** structure. (See **CFDDEV**, following, for the definition of this structure.)

If an error occurs, **errno** may be set to one of the following:

EEXIST Duplicate IODN entry in block I/O device table
ENODEV IODN not in block I/O device table
EINVAL No more room in block I/O device table.

Note: When **EINVAL** is returned, the value of the keyword **maxbios** (found in **/etc/master**), should be changed to a larger number and the kernel rebuilt.

CFDCODE Issues a **Define_Code** SVC to the VRM. The parameter is the following structure:

```
struct defcode {
    int  iocn; /* IOCN to define */
    int  opts; /* options word */
    int  ciocn; /* IOCN to copy */
};
```

The options available are:

ADD-IOCN Add IOCN option
DEL-IOCN Delete IOCN option
DUP-IOCN Duplicate IOCN option.

If the option specifies deleting or duplicating a module, the action is performed immediately. To add a module, a single **write** system call must immediately follow with the contents of the **a.out** module. The VRM uses the **write** buffer directly. The **write** buffer is cleared when it is returned from a call. The module must be aligned on a 2K page boundary and not mixed with other data.

CFDDEV Configures a device in the VRM that is not a disk partition. The parameter is a pointer to the following structure:

```
struct defdev {
    unsigned short  iodn; /* IODN to use */
    unsigned short  iocn; /* IOCN to use */
    unsigned short  opts; /* add/delete */
    unsigned short  chars; /* device characteristics */
    char  name[4]; /* device name */
    int  spare;
    union {
        struct {
            int  offhc; /* offset to hdw characteristics */
            int  offdc; /* offset to dev characteristics */
            int  offras; /* offset to RAS info */
        } offsets
        int  ddi[1]; /* device dependent info */
    } ddi_data;
};
```


The options available are:

ADD_IODN Add IODN option

DEL_IODN Delete IODN option.

A **Define_Device** SVC is issued to the VRM using the given data. The value is the Virtual Machine Interface (VMI) return code.

CFQDEV Issues a **Query_Device** SVC to the VRM. The parameter is a pointer to a structure of the following form:

```
struct qdev {
    unsigned short iodn;      /* IODN to query */
    unsigned short options;   /* Query device options */
    int length;               /* length of the following */
    char buffer[];            /* info returned here */
};
```

The options available are:

Q_HRDW Query hardware information

Q_DEV Query device information

Q_RAS Query RAS information.

After verifying the address and length of the structure, a **Query_Device** SVC is issued to the VRM with the structure given. The value returned is the value returned from the **Query_Device** SVC to the VRM.

CFRSTAT Returns the status code from the last SVC to the VRM issued from this driver. The parameter is ignored.

CFUDRV Configures an AIX kernel device driver. The parameter is a pointer to the following structure:

```
struct unxdrv {
    dev_t devno; /* major/minor device number */
    unsigned short iodn; /* IODN to set */
    unsigned short ddilen; /* device dep. info byte length */
    unsigned short lev; /* interrupt level */
    union { /* optional device dependent info */
        . . .
    };
};
```

config

```
struct {      /* for Send_Command calls */
    int rv2,
        rv3,
        rv4,
        rv5,
        rv6,
} ddi_sc;
} ddi;
};
```

The device initialization routine for the given driver is called with the given **iodn** and device-dependent information. Some device drivers may interpret the **lev** field as the VMI interrupt level to use. (Sublevels are always assigned dynamically by the kernel.) The value of the **CFUDRV ioctl** system call is the value returned by the device driver initialization routine. A successful call returns a value of 0; otherwise it returns a value of -1.

The device driver may modify the device-dependent information that is copied back into the structure provided by the caller of the **config** driver.

If both the **ddilen** and **iodn** fields are 0, the device initialization routine turns off the given minor number so that future calls to open that device will fail. If the **iodn** field is 0 and the **ddilen** field is a value other than 0, the device driver may perform various operations not directly relating to the minor device specified in the **devno** field. Device drivers associated with device managers issue a **Send_Command** SVC to the VRM device manager with the given values **rv2**, **rv3**, **rv4**, **rv5**, and **rv6** in registers 2, 3, 4, 5, and 6. The driver waits for the manager to return a completion or error interrupt and overwrites the **rv2**, **rv3**, and **rv4** words in the structure with data words 1, 2, and 3 returned from the interrupt. The value of the initialization routine is the status code returned with the interrupt. This can be obtained by issuing the **CFRSTAT ioctl** call previously mentioned.

CFUVRM Updates the VRM. The kernel issues an **Update_VRM** SVC to the VRM. The return value is the return code from the VMI call.

File

/dev/config

Related Information

In this book: “ioctl” on page 2-407, “fd” on page 5-33, “hd” on page 5-36 and “hft” on page 5-39.

The **vrconfig** command in *AIX Operating System Commands Reference*.

dft, . . .

dft, bsc

Purpose

Provides 3270 Distributed Function Terminal (DFT) and Binary Synchronous Communications (BSC) capabilities.

Synopsis

```
#include <sys/3270.h >
```

Description

The 3270 kernel device driver is a multiplexed device driver that supports an independent logical 3270 session on each of its channels. It can access multiple VRM device drivers, depending on the special file name used.

Open

The **open** system call initializes a path to a host session. The *oflag* parameter to **open** should be set to **O_RDWR**; all other values are ignored.

The specific adapter and port to be used is specified by the path name that is passed to **open**. The special files named **/dev/dftn** use a 3278/79 Emulation Adapter. The special files named **/dev/bscn** use the Logical Link Control from NETWORK 3270-PLUS (BSC), which uses a Multiprotocol Adapter. To use any **/dev/bscn**, you must have NETWORK 3270-PLUS (BSC) or NETWORK RJE-PLUS (BSC) installed on your system. More than one 3278/79 Emulation Adapter or Multiprotocol Adapter can be installed in your system.

A specific session on one of the ports can be addressed by following the special file name with a slash and the device address. For example, **/dev/bsc2/5** identifies device address 5 on a specific port of one of the Multiprotocol Adapters.

Note: It is possible to specify slow device, and printer configuration on a session by session basis. For information on slow device and printer configuration, see “ddi” on page 3-47 in this book. The path name specified to the **open** system call can specify this device address or not; if it is not specified, then an available device address is used.

The exact correlation between special file names and specific Multiprotocol Adapter ports is established when NETWORK 3270-PLUS (BSC) is installed.

Device address 0 is used to provide profile information for the NETWORK 3270-PLUS (BSC) Logical Link Control and to establish the connection. This applies to BSC only.

Reading and Writing

The *ext* parameter of the **readx** and **writex** extended I/O system calls is used as a pointer to an **io3270** structure that contains additional information. However, some simple applications may not need to supply the *ext* parameter and can use the **read** and **write** system calls.

The **io3270** structure is defined in the **sys/3270.h** header file, and it contains the following members:

```
uint io_flags; /* Information Flags */
uint status;  /* Status Codes      */
```

The value of **io_flags** is formed by logically ORing values from the following list:

- | | |
|--------------|---|
| LOCK | Lock or unlock: Lock is set when a write or writex system call is issued. It is reset (unlocked) when ready for the next write operation. |
| DATA | Data is available to be read. |
| TRANS | Transparent: Indicates that this buffer should be sent in transparent mode, or that it was received in transparent mode. This bit applies only to NETWORK RJE-PLUS (BSC). |
| COMM | Communication check: A communication error was detected. |
| PROG | Program check: A program error was detected. |
| MACH | Machine check: A machine error was detected. |

Notes:

1. The application read and write buffers are formatted as 3270 data streams.
2. A **write** or **writex** request fails if the adapter is currently sending outbound data, or if the host issued anything other than a READ BUFFER 3270 command in response to an ATTENTION sent by the VRM device driver.
3. Use the **SND_STATUS ioctl** operation to send status to the host, not **write** or **writex**.

select Support

The 3270 device driver supports the **select** system call in the following manner:

- Read selects are satisfied when input data is available.
- Write selects are always satisfied immediately.
- Exception selects are never satisfied, or hang indefinitely if no *timeout* value is specified.

See “select” on page 2-703 for more information about this system call.

ioctl Operations

The 3270 device driver performs the following **ioctl** operations. See “**ioctl**” on page 2-407 for a complete description of the **ioctl** system call.

int ioctl (fildes, IOCTYPE)

int fildes;

Returns a character that identifies the device type. This character is the value given for the **appt** keyword in the **/etc/system** file.

int ioctl (fildes, IOCINFO, arg)

int fildes;

struct devinfo *arg;

Stores information about the device into the **devinfo** structure, which is defined in the **sys/devinfo.h** header file.

int ioctl (fildes, GET-STATUS, arg)

int fildes;

struct io3270 *arg;

Gets the device driver status and stores it in the **io3270** structure pointed to by the **arg** parameter. See “Reading and Writing” on page 5-27 for a description of the **io3270** structure.

int ioctl (fildes, SND-STATUS, arg)

int fildes;

struct nsddsstat *arg;

Sends status and sense data to the host, as specified in the structure pointed to by the **arg** parameter. The **nsddsstat** structure is defined in the **sys/3270.h** header file, and it contains the following members:

```
struct code    co_de
struct status  sta_tus
```

The **code** structure contains the following members:

```
unsigned bit0 : 1;
unsigned bit1 : 1;
unsigned db   : 1;    /* Device Busy */
unsigned us   : 1;
unsigned de   : 1;    /* Device End  */
```


The **status** structure contains the following members:

```

unsigned bit0 : 1;
unsigned bit1 : 1;
unsigned cr   : 1;
unsigned ir   : 1;    /* Intervention Required */
unsigned ec   : 1;
unsigned dc   : 1;
unsigned oc   : 1;

```

Diagnostics

Systems calls to **dft** fail and set **errno** to indicate the error if one or more of the following are true:

EINSUFRES	100	Insufficient resources.
EATTACH	101	Device already attached.
EMAXATTACH	102	Device already attached to the maximum number of machines.
EADDALIGN	103	GPR4 is not word-aligned.
EIODN	104	Invalid iodn .
EINVINTR	105	Invalid inter level or sublevel.
EINTRMAX	106	Invalid inter maximum.
EOPCANCEL	107	Operation canceled.
EMEM	108	Insufficient memory or page pinned 255 times.
EINVCMDSEG	109	Invalid command extension segment ID.
EINVCMDEXT	110	Invalid length or alignment error
EPATHID	111	Invalid path identifier.
EOPOP	112	Reserved bit set in Operation Options.
EOPREJECT	113	Operation rejected.
EOPERATION	114	Error during operation.
ENSDDCORR	115	The nsdd ring correlators do not match.
EINVCCB	116	Invalid ccb or buffer segment identifier.
EINVLENGTH	117	Length specification error.
EINDEVICE	118	Invalid device or media data.
EINVOPTION	119	Invalid operation.

Files

<code>/dev/dft0</code>	First 3278/79 Emulation Adapter
<code>/dev/dft1</code>	Second 3278/79 Emulation Adapter
<code>/dev/dft2</code>	Third 3278/79 Emulation Adapter
<code>/dev/dft3</code>	Fourth 3278/79 Emulation Adapter
<code>/dev/bsc0</code>	Identifies a Multiprotocol Adapter port to the BSC LLC
<code>/dev/bsc1</code>	Identifies a Multiprotocol Adapter port to the BSC LLC
<code>/dev/bsc2</code>	Identifies a Multiprotocol Adapter port to the BSC LLC
<code>/dev/bsc3</code>	Identifies a Multiprotocol Adapter port to the BSC LLC.

Related Information

In this book: “ioctl” on page 2-407, “read, readx” on page 2-591, “write, writex” on page 2-876, and “devinfo” on page 3-66.

NETWORK 3270-PLUS (BSC) User Guide.

NETWORK RJE-PLUS (BSC) User Guide.

error

Purpose

Logs system events.

Synopsis

```
#include <sys/err.h>
#include <sys/erec.h>
```

Description

The format of an event record depends on the type of event encountered. Each record, however, has a header with the following format:

```
struct errhdr {
    unsigned e_len;      /* word in record (with header) */
    time_t   e_time;     /* time of day */
    long     e_timex;    /* clock ticks */
    char     e_nid[8];   /* node ID */
    char     e_vmid[8];  /* virtual machine ID */
    union {
        struct {
            char ex_class;
            char ex_subclass[2];
            char ex_type; /* record type */
        } ex;
        int csmt;
    } exx;
};
```

The error daemon searches the RAS configuration file **/etc/rasconf** for a stanza labeled **/dev/error**. Minor device 0 of the **error** driver is the interface between a process and the routines that collect error records in the system. This driver can be opened only for reading by a process (usually the error daemon) with superuser permission. Each read retrieves an entire error record. A read request of less than the entire record causes the retrieved record to be truncated. Multiple processes can open the **error** file to write.

error

File

`/dev/error`

Related Information

In this book: “errunix” on page 2-224, and “rasconf” on page 3-189.

The **errdemon** command in *AIX Operating System Commands Reference*.

Device Driver Development Guide.

fd

Purpose

Supports the diskette device driver.

Synopsis

```
#include <sys/devinfo.h>
```

Description

The diskette special file provides block and character (raw) access to diskettes in the diskette drives, allowing only one process to have a diskette drive open for writing at a time. The **config** device driver associates the minor device number with a particular diskette drive. Normally, the special file **/dev/fdn** is given the minor device number *n*. Removing the diskette from the drive with diskette files still open can cause various I/O system calls to return errors.

The minor device number specifies both the drive number and the format of the diskette to be read or written. Assume that **/dev/fdn** corresponds to a diskette drive with minor device number *n*. In this case, **fd0**, **fd1**, **fd2**, and **fd3** specify diskette drives 0 through 3, respectively, without specifying their format.

Using **fs0**, . . . , **fs3**, which correspond to minor device numbers 4 through 7, forces a diskette to be treated as a single-sided diskette. Similarly, **fd0.8**, . . . , **fd3.8**, which correspond to minor device numbers 8 through 11, force the diskette to be treated as an 8-sectored diskette. **fs0.8**, . . . , **fs3.8**, which correspond to minor device numbers 12 through 15, force the diskette to be treated as single-sided and 8-sectored.

Configuration Data

The **config** device driver is called during system initialization to customize diskettes. This is accomplished by calling the device driver at its initialization entry. For diskettes, no device-dependent information is required, so the information for customizing diskettes has the following structure:

```
struct {  
    dev_t devno;           /* major/minor device number */  
    unsigned short iodn;   /* IODN to set */  
    unsigned short ddilen; /* device dependent info length */  
    unsigned short lev;    /* ignored */  
};
```

```
        union {                                /* optional device dependent info */
            char ddi_fd[];
            . . .                                /* ddi for other devices */
        } ddi;
    };
```

ioctl Operations

The **IOCTYPE** **ioctl** system call returns the device type DD-DISK, defined in the **sys/devinfo.h** header file.

The **IOCINFO** **ioctl** system call returns the following structure, defined in the **sys/devinfo.h** header file:

```
struct devinfo {
    char devtype;
    char flags;
    union {
        struct {                                /* for disks */
            short bytpsec; /* bytes per sector */
            short secptrk; /* sectors per track */
            short trkpcyl; /* tracks per cylinder */
            long numblks; /* number of blocks on diskette */
        } dk;
        . . .                                /* for other devices */
    } un;
};

/* flags */
#define DF_FIXED 01 /* non-removable */
#define DF_RAND 02 /* random access possible */
#define DF_FAST 04 /* a relative term */
```

Files

```
/dev/fd0, /dev/fd1, . . .
/dev/rfd0, /dev/rfd1, . . .
```


Related Information

In this book: “config” on page 5-21, “fs” on page 3-100, and “ioctl” on page 2-407.

hd

hd

Purpose

Supports the fixed-disk device driver.

Synopsis

```
#include <sys/devinfo.h>
```

Description

The fixed-disk device driver provides block and character (raw) access to minidisks on the fixed-disk drives. The **config** device driver associates the minor device number with the minidisk. Normally, the special files **/dev/hdn** and **/dev/rhdn** is given the minor device number *n*.

The minidisk with minor device number 0 is always the minidisk used to initially load the system program.

In raw I/O , the buffer must always begin on a fullword boundary, and counts should be a multiple of 512 bytes (a disk block). Likewise **lseek** system calls should specify a multiple of 512 bytes.

Configuration Data

The **config** device driver is called at its initialization entry point during system initialization to customize minidisks. The following shows the structure of the customize information:

```
struct {
    dev_t    devno;           /* major/minor device number */
    unsigned short iodn;      /* IODN set */
    unsigned short ddilen;    /* device dependent info length */
    short    lev;             /* ignored */
}
```

```

union {
    struct {
        int rv2,
        rv3,
        rv4,
        rv5,
        rv6;
    } ddi_sc;
    . . .
} ddi;
};

```

If the **iodn** field is 0, the manager receives a **Send-Command** supervisor call (SVC) with values **rv2**, **rv3**, **rv4**, **rv5** and **rv6** in registers 2, 3, 4, 5, and 6. The driver waits for the manager to return a completion or error interrupt. The return value is the status code returned at the interrupt, which can be obtained using an **ioctl** system call to the **config** device driver.

ioctl Operations

The **VQUERY ioctl** call queries the disk write-verify status for the minidisk. It uses the form:

```

ioctl (fildes, command, arg)
struct wverify *arg

```

where **arg** is a pointer to the one-word structure **wverify** that contains the write-verify status to be returned. A value of 1 returned indicates enabled and 0 if disabled.

The **VCNTRL ioctl** call enables or disables write-verify for the minidisk. It uses the form:

```

ioctl (fildes, command, arg)
int arg;

```

where an **arg** value of 0 is used to disable disk write-verify and a value of 1 is used to enable disk write-verify.

See "mdverify" on page 2-443 for another way to set and query the write-verify status of a minidisk.

The **IOCTYPE ioctl** call returns the value **DD-DISK**, defined in **<sys/devinfo.h>**.

The **IOCINFO ioctl** call returns the following structure, defined in **<sys/devinfo.h>**:

```

struct devinfo {
    char devtype;
    char flags;

```


hd

```
    union {
        struct {          /* for disks */
            short bytpsec; /* bytes per sector */
            short secptrk; /* sectors per track */
            short trkpcyl; /* tracks per cylinder */
            long numblks;  /* blocks this mini-disk */
        } dk;
        . . .              /* for other devices */
    } un;
};

/*flags */
#define DF_FIXED 01 /* non-removable */
#define DF_RAND  02 /* random access possible */
#define DF_FAST  04 /* a relative term */
```

Files

```
/dev/hd0, /dev/hd1, . . .
/dev/rhd0, /dev/rhd1, . . .
```

Related Information

In this book: “config” on page 5-21, “fs” on page 3-100, “ioctl” on page 2-407, and “lseek” on page 2-434.

hft

Purpose

Implements a high-function virtual terminal device.

Synopsis

```
#include <sys/hft.h>
```

Description

The **hft** device driver supports a virtual terminal concept based on the virtual terminal subsystem of the Virtual Resource Manager. The following information is intended to supplement the discussion of the virtual terminal subsystem in *VRM Programming Support*. Additional information can also be found in the file `/usr/lib/samples/README.hft`.

The virtual terminal concept supports the illusion that more devices exist than are physically present and that these devices have characteristics and features not necessarily limited by the actual devices. In addition to displays and keyboards, virtual terminals support locators, valuator, lighted programmable function keys, and sound generators. Virtual terminals are logically independent of each other but share physical resources over time. The virtual terminal that can accept physical input at a given time is called the **active** virtual terminal.

The virtual terminal provides a model of a single terminal that can be in one of the following modes at a given time:

- Keyboard Send-Receive mode (KSR)
- Monitor mode (MOM).

The KSR mode emulates an ASCII terminal using an RT ASCII data stream, which is described in detail in “data stream” on page 4-5. The monitor mode allows applications to have a direct output path to the display hardware and shortened path for input. The form of the data accepted in each mode is unique to that mode. This optimizes the movement of data between the virtual terminal and the application program and supports the different functions within each mode. The default mode is KSR, which supports existing applications expecting an ASCII terminal.

The virtual terminal supplies default values for keyboard-to-character mapping, character-to-display mapping, echo/break specification, tab rack, and protocol mode flags to be used until a redefinition is received from the application.

This **hft** facility is the kernel-level support for virtual terminals. Since the association of virtual terminals to physical terminals is dynamic, this special file, which represents the physical terminal, is multiplexed across virtual terminals by expanding the **open**, **close**, **read**, **write**, and especially the **ioctl** system calls to the driver. This type of driver is specified by the **S_IMPX** bit in the **stat.h** file. Many extra **ioctl** system calls are provided to allow access to advanced features of the **hft** facility. The facilities described in “termio” on page 5-133 also apply to the virtual terminal.

The first (or only) **hft** is known as minor device 0, and special file **/dev/hft** is associated with it. The special file **/dev/console** is minor device 1. Minor devices 2 and higher are associated with additional **hft** physical terminals, if there are any.

Each time **/dev/hft** is opened, a new **hft** virtual terminal is created and opened. A maximum of 16 virtual terminals can be opened due to limits on system resources.

To reopen an existing virtual terminal, open the special file **/dev/hft/i**, where *i* is the number of an open driver channel. The channel number can be determined with the **HFGCHAN ioctl** operation. The **/dev/console** special file is channel number 1.

A process can also communicate with the **hft** screen manager by opening the **/dev/hft/mgr** file. Only the screen manager **HFQSMGR** and **HFCSMGR ioctl** operations can be issued to this file. The **read** and **write** system calls are not allowed.

The **/usr/lib/samples/hft** directory contains sample programs that use the **hft** virtual terminal subsystem. See the file **/usr/lib/samples/README.hft** for more information about these sample programs. cp 2i

The following table of contents can be used to locate specific topics:

Contents of hft Section

Initial State	5-43
termio Support	5-44
select Support	5-45
ioctl Operations	5-46
Query I/O Error (HFQEIO)	5-46
Query Device (HFQDEV)	5-46
Reconfigure (HFRCONF)	5-48
Get Channel Number (HFGCHAN)	5-51
Set Echo and Break Maps (HFSECHO)	5-51
Set Keyboard Map (HFSKBD)	5-53
Set Japanese Keyboard Map (HFSJKBD)	5-56
Get Virtual Terminal ID (HFGETID)	5-57
Query (HFQUERY)	5-58
Query Device IDs Command	5-58
Query Physical Device Command	5-59
Query Locator Command	5-63
Query LPFKs Command	5-64
Query Dials Command	5-65
Query Presentation Space Command	5-65
Query HFT Device Command	5-66
Query DMA Command	5-67
Enable Sound Signal (HFESOUND)	5-67
Disable Sound Signal (HFDSOUND)	5-68
Enter Monitor Mode (HFSMON)	5-68
Exit Monitor Mode (HFCMON)	5-68
Query Screen Manager (HFQSMGR)	5-69
Control Screen Manager (HFCSMGR)	5-70
DMA Move (HFMDMA)	5-72
Input	5-73
Untranslated Key Control	5-73
Input Device Report	5-74
Adapter-Generated Input	5-76
General Output	5-78
Protocol Modes	5-79
Set Keyboard LEDs	5-81
Set Locator Thresholds	5-81
Set Tablet Dead Zones	5-82
Set LPFKs	5-82

Set Dial Granularities	5-83
Sound	5-83
Cancel Sound	5-84
Change Physical Display	5-84
 Keyboard Send-Receive Mode (KSR) Output	 5-86
Character Set Definition	5-86
Set KSR Color Palette	5-87
Change Fonts	5-87
Cursor Representation	5-88
 Monitor Mode (MOM) Output	 5-90
Entering Monitor Mode	5-90
Screen Request and Input Ring Buffer Definition	5-90
Reading Input Data from the Ring Buffer	5-92
Next Window Function	5-93
Exiting Monitor Mode	5-93
Signals	5-94
 Considerations for hft Emulation	 5-95

Initial State

When a new terminal is opened, it is initialized to a known state. This initial state can be changed, if desired. The initial terminal state is the following:

- Mode: Keyboard Send-Receive (KSR).
- Echo/Break Map: Echo all characters; break for none.
- Tab Rack: The first, every eighth, and the last position of every line.
- ASCII Controls:

LNM	Set
IRM	Not set
SRM	Not set
TSM	Not set
CLM	Not set
AUTONL	Set.

- Protocol Mode:

HFWRAP	Set
HFHOSTPC	Not set
HFXLATKBD	Set
HFHOSTS	Not set
HFLPFKS	Not set
HFDIALS	Not set
HFDINTRONLY	Not set
HFDINTR	Not set
HFJKANA	Not set.

See "Protocol Modes" on page 5-79 for a complete explanation of the protocol mode settings.

- Locator Threshold: 2.75 millimeters horizontal, 5.5 millimeters vertical.
- Font: Initially, and whenever the physical display device is changed, this is set to be the first font in the customized list of fonts that:
 - Results in a presentation space of 80 columns by 25 rows, and
 - Has a normal appearance (not bold or italic).

If no font meets these criteria, then the first font that can be displayed on the device is chosen. All alternate fonts are initialized to the selected font.

- Character mode color palette for both foreground and background:

Entry	Color
0	Black
1	Red

2	Green
3	Yellow
4	Blue
5	Magenta
6	Cyan
7	White
8	Gray
9	Light red
10	Light green
11	Brown
12	Light blue
13	Light magenta
14	Light cyan
15	High intensity white.

termio Support

This section briefly describes the input, output, and line discipline modes and the **ioctl** system calls from **termio** that are supported by **hft**. For more detail, see “termio” on page 5-133.

The following list identifies input modes supported by **hft**:

INLCR	
IGNCR	
IUCLC	
IXANY	
ICRNL	Supported by using the keyboard remap facility to change the code sent by the Enter (Return) and Ctrl-M keys.
IXON	If Ctrl-S is pressed while output is being performed on the screen, the output does not stop until the end of the current write system call.

The following list contains the output modes supported by **hft**.

OPOST
ONLCR
OCRNL

The supported basic and enhanced edit line discipline modes are:

ISIG
ICANON
ECHO
ECHOE
ECHOK
ECHONL
NOFLSH

Terminal paging is supported using the **TCGLEN** and **TCSLEN** **ioctl** operations. When paging is active, the contents of the buffer supplied by the **write** system call are written out in page-size pieces as specified by **TCGLEN** and **TCSLEN**.

Other **termio** operations supported by **hft** include the **ioctl** commands **TCXONC** and **TCFLSH**.

select Support

The **hft** device driver supports the **select** system call in the following manner:

- Read selects are satisfied when input data is available.
- Write selects are always satisfied immediately.
- Exception selects are never satisfied, or hang indefinitely if no *timeout* value is specified.

See “select” on page 2-703 for more information about this system call.

ioctl Operations

The **hft** supports a number of operations issued by the **ioctl** system call to provide access to sophisticated features of the **hft**. See “**ioctl**” on page 2-407 for details about the syntax of the system call itself. For information about issuing requests for these operations to an emulated **hft** device, see “Considerations for hft Emulation” on page 5-95.

Query I/O Error (HFQEIO)

The HFQEIO operation returns detailed device error code information. If an I/O operation or other system call to the **hft** fails due to a hardware error, the system returns a nonzero value and sets **errno** to **EIO**. The calling program can get a more detailed device error code by using the **ioctl** system call to issue an HFQEIO operation. The HFQEIO operation is invoked by the following:

```
int ioctl(fildes, HFQEIO, 0)
int fildes;
```

The return value from the HFQEIO **ioctl** operation is either 0 (indicating that the last I/O operation was successful), -1 (indicating that the HFQEIO operation itself failed), or the error code for the last **hft** I/O operation. See *VRM Device Support* for an explanation of the individual virtual terminal error codes.

Query Device (HFQDEV)

The HFQDEV operation obtains detailed device information about the device types associated with the virtual terminal. For details about this query operation, see “Query Device SVC” in *VRM Programming Support*. The HFQDEV operation is invoked by the following:

```
int ioctl(fildes, HFQDEV, arg)
int fildes;
struct hfqdev *arg;

struct hfqdev
{
    unsigned short hf_qdrsvd;
    unsigned short hf_qdopts;
    unsigned int   hf_qdlen;
};
```


This **ioctl** operation stores information into a **hfqdrsp** structure. The **hfqdrsp** structure overlays the **hfqdev** structure in memory. The **hf_qdopts** field is the only option recognized and must be set to the value 2. The **hfqdrsp** structure contains the following fields and values:

Field	Description
hf_vtrmiodn	The virtual terminal resource manager IODN.
hf_vtrmiocn	The virtual terminal resource manager IOCN.
hf_devtype	The device type. The value in this field is 0x0002 (shared device).
hf_devname[4]	The device name. The value is VTRM for Virtual Terminal Resource Manager.
hf_hwoffset	The offset to hardware characteristics.
hf_devoffset	The offset to the device characteristics.
hf_erroffset	The offset to the error log.

The next eight fields contain information about the last operation completed by the virtual terminal. For more detailed information about these fields, see the discussion of the Query Device SVC in *VRM Programming Support*.

hf_newics	New Interrupt Control Status register value.
hf_statflags	Status flags.
hf_ovrncnt	Overflow count.
hf_opresult	Operation result.
hf_deviocn	Device IOCN.
hf_dataword1	Device-dependent data or command extension segment ID.
hf_dataword2	Device-dependent data or command extension address.
hf_dataword3	Device-dependent data.
hf_ddilen	The length (in words) of the device-dependent information.
hf_rc	The return code from IPL or the Define-Device SVC. A value of 0 indicates a successful operation.
hf_smiocn	The screen manager IOCN.
hf_vtmiocn	The virtual terminal mode processor IOCN.
hf_keyiocn	The keyboard IODN.
hf_lociocn	The locator IODN. This field is set at IPL time. Note that the locator is optional. If the locator is not used, this field is set to 0.

hf-spkiodn	The speaker IODN. This field is set at IPL time.
hf-numfont	The number of fonts. One font is supplied by the VRM; however up to 31 additional fonts can be defined.
hf-fontiocn[i]	The font IOCNS. Undefined font IOCNS fields are set to 0.
hf-numdisp	The number of physical displays. This field is completed at IPL time. The VRM supports as many as four physical displays.

The next three fields (**hf-devid**, **hf-deviodn**, and **hf-deviocn**) are repeated four times to accommodate additional displays. Fields in this array that are not used are set to 0.

hf-physd[i].hf-devid

Contains a value that is a code identifier for a particular physical device in use, such as a display adapter or monitor combination.

hf-physd[i].hf-deviodn

The IODN for the physical display device.

hf-physd[i].hf-deviocn

The IOCNS for the virtual display driver.

hf-keymapiocn The IOCNS that defines how key positions map to characters.

hf-chrmapiocn The IOCNS that defines how characters map to display codes for the Unique 1 and Unique 2 character sets.

hf-echomapiocn The IOCNS that defines the echo and break maps.

hf-initiocn The IOCNS of the virtual terminal mode processor initialization parameters. Examples of these parameters include protocol modes, tab rack, and so on.

hf-dialsiodn The IODN of the valuator dial device driver.

hf-lpfkiodn The IODN of the lighted program function key device driver.

Reconfigure (HFRCONF)

The HFRCONF operation allows a user program to reconfigure the virtual terminal to include different real devices. Helpful information about virtual terminal reconfiguration can be found in the file `/usr/lib/samples/README.conf`. This operation is invoked by the following:

```
int ioctl(fildes, HFRCONF, arg)
    int fildes;
    struct hfrconf *arg;

    struct hfrconf
```



```

{
    unsigned hf_op;
    unsigned hf_obj;
    union
    {
        uint hf_infob;
        struct
        {
            ushort hf_iodn;
            ushort hf_iocn;
        } hf_info2;
    } hf_info;
};

```

This command changes the configuration of the physical terminal or the virtual terminal defaults.

The **hf-op** field contains the requested operation. The valid operations appear in the following list. These reconfiguration operations, with the exception of those followed by an * (asterisk), take effect only for terminals opened after the reconfiguration. The operations followed by an asterisk take effect for the terminals that are currently open as well as those opened after the reconfiguration.

- | | |
|----------------------|---|
| HFADDLOC | Adds a real locator. hf-obj contains the real locator device driver IODN. |
| HFADDSOUND | Adds a real sound device. hf-obj contains the real sound device driver IODN. |
| HFADDDISPLAY | Adds a real display. The following fields must also be set for this operation:
<div style="margin-left: 2em;"> hf-obj The real display identifier
 hf-iodn The display device driver IODN
 hf-iocn The virtual display driver IOCN. </div> |
| HFDELDISPLAY | Deletes a real display. hf-obj contains the real display identifier. |
| HFADDFONT | Adds a font. hf-obj contains the font IOCN. |
| HFCHGKBDRATE* | Changes the keyboard typematic rate. Bits 24 - 31 of hf-obj indicate the keyboard typematic rate. For the standard RT keyboard, valid values are between 2 and 40 characters per second and can be incremented in 1 character-per-second units. The default value for the RT keyboard is 14 characters per second. |

HFCHGKBDEL*	Changes the keyboard typematic delay. Bits 16 – 31 of hf_obj indicate the keyboard typematic delay. For the standard RT keyboard, valid values are between 300 and 600 milliseconds and can be incremented in 100 millisecond units. The default value for the RT keyboard is 400 milliseconds.
HFCHGLOCRATE*	Change locator sample rate. Bits 24 – 31 of hf_obj indicate the locator sample rate. For the standard RT locator, valid values are 10, 20, 40, 60, 80, and 100 samples per second. The default for the RT locator is 60 samples per second.
HFCHGCLICK*	Turns the keyboard click mechanism on or off. Bit 31 of hf_obj indicates whether the speaker produces a sound when a key is pressed. Sound is suppressed when bit 31 equals 0 and produced when bit 31 equals 1. The default for the RT keyboard is keyboard click on.
HFCHGVOLUME*	Sets the sound volume level. Bits 24 – 31 indicate the volume of sounds produced by the speaker. For the standard RT speaker, valid values are 0 (sound off) and, 1 (low volume), 2 (medium volume) and 3 (high volume). The default for the RT speaker is medium volume.
HFKEYMAP	Replaces the position code map. hf_obj contains the new position code map IOCN. See the <code>/usr/lib/samples/hft/hftkbdmap.c</code> file.
HFDISPMAP	Replaces the character code maps for the Unique 1 and Unique 2 character sets. See the <code>/usr/lib/samples/hft/hftchrmap.c</code> file. hf_obj contains the new unique character code map IOCN.
HFECHOMAP	Replaces the echo/break map. hf_obj contains the new echo/break map IOCN. See the <code>/usr/lib/samples/hft/hftecbmap.c</code> file.
HFDEFAULT	Replaces miscellaneous default values. hf_obj contains the new miscellaneous defaults IOCN. See the <code>/usr/lib/samples/hft/hftmiscdef.c</code> file.
HFSETDD	Changes the default display. hf_obj contains the real display identifier.
HFADDDIALS	Adds a real dial device. hf_obj contains the dial device driver IODN.
HFADDLPFK	Adds a real lighted programmable function key (LPFK) device. hf_obj contains the LPFK device driver IODN.
HFCHNGDMA	Changes the DMA start address and length. hf_obj contains the new DMA start address, and hf_infob contains the length of the new DMA area.

Get Channel Number (HFGCHAN)

The HFGCHAN operation returns the current driver channel number as the value of the `ioctl` system call. This number can be used to open a specific virtual terminal. The `arg` parameter is ignored. The HFGCHAN operation is invoked by the following:

```
int ioctl(fildes, HFGCHAN, 0)
int fildes;
```

Set Echo and Break Maps (HFSECHO)

The HFSECHO operation sets the **hft** echo and break maps. **Echoing** displays the character associated with a keystroke on the screen or performs the function associated with a control. **Breaking** switches the input path from the monitor mode input buffer to the unsolicited ASCII data stream flow. Echoing applies only to KSR mode; breaking applies only to MOM mode. Echoing and breaking can be selectively enabled for each ASCII code point and multibyte control sequence. The default is to echo all characters and control sequences, but not to break on any of them.

The HFSECHO operation is invoked by the following `ioctl` call:

```
int ioctl(fildes, HFSECHO, arg)
int fildes;
struct hfbuf *arg;

struct hfbuf
{
    char *hf_bufp;
    int hf_buflen;
};
```

The **hf_bufp** field points to an array of 32 integers. The **hf_buflen** field contains the value 128 (0x80), which is the length of the array in bytes. The first 16 integers constitute the echo map; the second 16 integers are the break map.

Each of the two maps is treated as a set of bits. Bit 0 is the most significant bit of the first integer. Bit 511 is the least significant bit of the sixteenth integer. Each bit corresponds to an ASCII code point or multibyte control. Bits 0 through 255 (0xFF) correspond to the single-byte codes. Bits 256 (0x100) and higher correspond to multibyte control sequences, as illustrated in Figure 5-1 on page 5-52. Bit 511 (0x1FF) specifies whether to echo or break on invalid and unsupported multibyte control sequences. See “data stream” on page 4-5 for a detailed explanation of each of the multibyte control sequences.

Hex2	←Most Significant Hex Digits 0,1→							
	10		12	13	14	.	.	1F
0	CBT	DMI		RC				
1	CHA	EMI	RI	KSI				
2	CHT	EA		VTD				
3	CTC	ED	RIS					
4	CNL	EF	RM					
5	CPL	EL	SD					
6	CPR	ECH	SL					
7	CUB	GSM	SR					
8	CUD	HTS	SU					
9	CUF	HVP	SGR					
A	CUP	ICH	SG0					
B	CUU	IL	SG1					
C	CVT	IND	SM					
D	DCH	NEL	TBC					
E	DL	PFK	VTS					
F	DSR		SC					INV

Figure 5-1. Bit Positions of ASCII Controls in Echo Map

For the echo map, a bit set to 1 means the character or control sequence is echoed when a key that is mapped to it is pressed. The echo map is active only in KSR mode and can be set only from KSR mode.

For the break map, a bit set to 1 means that the character or control sequence is reported using the **read** system call instead of being placed in the input ring buffer. Also, the **SIGMSG** signal is sent to the process to indicate that input data is available. The break map is active only in monitor mode. (See “Monitor Mode (MOM) Output” on page 5-90 for a description of the input ring buffer.)

The echo and break maps are shared by all code pages. For P0 graphic code points (0x20 to 0xFF), bits 32 to 255 (0x20 to 0xFF) of each map are used. For other code pages, each half of the code page is associated with bits 128 to 255 (0x80 to 0xFF). For example, bit 160 (0xA0) specifies the echo or break status of code points P0 0xA0, P1 0x20, P1 0xA0, P2 0x20, and P2 0xA0.

Set Keyboard Map (HFSKBD)

The HFSKBD operation sets the keyboard map. Most keys on the keyboard can be remapped, changing the character or control sequence each key generates when pressed. See “keyboard” on page 5-97 for additional details. The HFSKBD operation is invoked by the following:

```
int ioctl(fildes, HFSKBD, arg)
    int fildes;
    struct hfbuf *arg;

    struct hfbuf
    {
        char *hf_bufp;
        int  hf_buflen;
    };
};
```

The **hf_bufp** field points to a **hfkeymap** structure and **hf_buflen** contains the length of the **hfkeymap**.

```
struct hfkeymap {
    char hf_rsvd1 ;
    char hf_nkeys;
    struct hfkey {
        char hf_kpos;
        char hf_kstate;
        struct hfkeyasgn
        {
            /* for single character */
            char hf_pagenum; /* Code page */
            char hf_character; /* Character to map */
        } hf_keyasgn;
    } hfkey[HFNKEYS];
};
```

};

The **hfkeymap** structure can remap one or more keys. The number of keys to remap is specified by the **hf_nkeys** field. One **hfkey** structure for each key specified in **hf_nkeys** follows. **HFNKEYS**, which is used as the dimension for the **hfkey** array, has a default value of 1, thus allowing one key to be remapped. To change **HFNKEYS**, set its value in a **#define** statement that comes before the **#include <hft.h>** statement.

The **hfkey** structure contains information for each key being remapped, such as key position, shift states, and the type of remapping being done. The fields and values in the **hfkey** structure are:

hf_kpos The key position number. See “keyboard” on page 5-97.

hf_kstate This field is subdivided into three groups of bits:

HFMAPMASK

Defines the bits that specify the type of mapping to be performed:

HFMAPCHAR	Specifies mapping a single character to a key.
HFMAPNONSP	Specifies mapping a nonspacing character to a key. (See “data stream” on page 4-5 for information about nonspacing characters.)
HFMAPFUNC	Specifies mapping a function ID to a key.
HFMAPSTR	Specifies mapping a string of more than one character to a key.

HFSHFMASK

Defines the bits that specify the shift state that applies to the key being mapped:

HFSHFNONE	Specifies the base state (no shift state)
HFSHFSHFT	Specifies the shift state
HFSHFCTRL	Specifies the Ctrl state
HFSHFALT	Specifies the Alt state
HFSHFALTGR	Specifies the Alt Gr (Alternate Graphics) state.

HFCAPSL

Specifies whether the **Caps Lock** state affects the key. If set, when **Caps Lock** mode is on, the base state of a key functions as the shift state, and the shift states functions as the base state.

The **hfkeyasgn** structure specifies the key to be remapped and the character codes generated when the key is pressed or released. The fields of this structure differ depending on the value of the **HFMAPMASK** bits in **hf_kstate**:

HFMAPCHAR, HFMAPNONSP:

hf_page	Specifies the code page
hf_char	Specifies a character (also called a code point) in that code page.

HFMAPSTR:

hf_page Specifies the code page
hf_kstrl Specifies *(the length of the string in bytes)* minus 1.

This is immediately followed by the string.

Note: Due to limitations of the **hfkeymap** structure, only one key can be assigned a string value. This key must be the last key specified in the **hfkey** array because the **hfkeymap** structure itself does not contain space for the variable-length string. This string must immediately follow the **hfkeymap** structure in memory. The virtual terminal subsystem supported by the VRM allows any number of keys to be assigned string values. To assign string values, set up your own key map buffer instead of using **hfkeymap**.

HFMAPFUNC:

hf_keyidh Specifies the high-order byte of the function ID.
hf_keyidl Specifies the low-order byte of the function ID.

The following list gives the function IDs for each of the functions that can be assigned to keys. See "Multibyte Controls" on page 4-13 for more details about these functions.

ID	Name
----	------

0x0000 — 0x00FE	(PFK) Issues the Programmable Function Key sequence for PF key 1 (ID = 0x0000) through 255 (ID = 0x00FE).
0x0101	(CUU) Moves the application cursor up one line.
0x0102	(CUD) Moves the application cursor down one line.
0x0103	(CUF) Moves the application cursor forward one character.
0x0104	(CUB) Moves the application cursor backward one character.
0x0105	(CBT) Moves the application cursor to the previous horizontal tab stop or beginning of field.
0x0106	(CHT) Moves the application cursor to the next horizontal tab stop or beginning of field.
0x0107	(CVT) Moves the application cursor down one vertical tab stop.
0x0108	(HOME) Moves the application cursor to the first line, first character in the presentation space.
0x0109	(LL) Moves the application cursor to the last line, first character in the presentation space.
0x010A	(END) Moves the application cursor to the last line, last character in the presentation space.
0x010B	(CPL) Moves the application cursor to the first character of the previous line.
0x010C	(CNL) Moves the application cursor to the first character of the next line.
0x0151	(DCH) Deletes the character over the application cursor.
0x0152	(IL) Inserts one line following the line of the application cursor.
0x0153	(DL) Deletes the line of the application cursor.
0x0154	(EEOL) Erases to the end of the line.

0x0155 (EEOF) Erases to the next tab stop.
0x0156 (CLEAR) Erases all characters from the presentation space.
0x0157 (INIT) Restores the initial state of the virtual terminal. (See the description of RIS in "Multibyte Controls" on page 4-13.)
0x0162 (RI) Performs one line reverse index control.
0x0163 (IND) Performs one line index control.
0x01FF (IGNORE) Sends no information when the key is pressed.

Note:

On the U.S. 101-key keyboard, the left **Alt** key produces the **Alt** shift state, and the right **Alt** key produces the **Alt Gr** shift state. The default keyboard mapping for the **Alt** and **Alt Gr** states is identical for all keys.

If a U.S. 101-key keyboard is attached, then mapping the **Alt** state of a key automatically causes the same mapping to be assigned to the **Alt Gr** state. This allows the two **Alt** keys on the U.S. keyboard to report the same ASCII codes. If you want to remap both the **Alt** and **Alt Gr** states of a key, you must remap the **Alt** state first, then the **Alt Gr** state. Software written primarily for keyboards other than the U.S. keyboard should remap the states in this order to ensure compatibility.

If the Japanese 106-key keyboard is attached, then access to the **Alt Gr** shift state is not possible.

Set Japanese Keyboard Map (HFSJKBD)

The HFSJKBD operation sets the Japanese keyboard map. This **ioctl** system call is designed to work with the 106-key Japanese keyboard and Japanese licensed program software only.

The HFSJKBD operation is invoked by the following **ioctl** system call.

```
int ioctl(fildes, HFSJKBD, arg)
```

The structures used in the HFSJKBD **ioctl** are the same as those in "Set Keyboard Map (HFSKBD)" on page 5-53 with the following difference in the interpretation of the shift states.

The Japanese keyboard has the following shift states:

- Romaji base
- Romaji shift
- Control
- Alternalte
- Kana base
- Kana shift.

For base and shift, the distinction between romaji and kana script systems is determined by the **Alt Gr** bit in the **hf_kstate** field of the **hfkey** structure. This bit can be combined with the base or shift bit to indicate one of four states. When this bit is 0, romaji is assumed. When this bit is 1, kana is assumed.

Note: The Japanese keyboard does not have a **Alt Gr** state.

The following literals are defined for this **ioctl**:

HFSHFMASK	Defines the bits that specify the shift state that applies to the key being mapped.
HFROMBASE	Specifies the base romaji state
HFROMSHFT	Specifies the shift romaji state
HFSHFCTRL	Specifies the Ctrl state
HFSHFALT	Specifies the Alt state
HFKANBASE	Specifies the base kana state
HFKANSHFT	Specifies the shift kana state.

Get Virtual Terminal ID (HFGETID)

The **HFGETID** operation gets identification information for the current **hft** virtual terminal. The **HFGETID** operation is invoked by the following:

```
int ioctl(fildes, HFGETID, arg)
    int fildes;
    struct hfgetid *arg;

    struct hfgetid {
        unsigned hf_iodn;
        unsigned hf_pgrp;
        unsigned hf_chan;
    };
```

The **hfgetid** structure contains the following fields and values:

Field	Value
hf_iodn	The I/O device number of the virtual terminal.
hf_pgrp	The process group ID; that is, the process ID of the terminal group leader.
hf_chan	The channel number. This channel number is also returned by the HFGCHAN ioctl operation.

Query (HFQUERY)

The HFQUERY operation gets information about the current virtual terminal. The HFQUERY operation is invoked by the following:

```
int ioctl(fildes, HFQUERY, arg)
    int fildes;
    struct hfquery *arg;

    struct hfquery {
        char *hf_cmd;
        int hf_cmdlen;
        char *hf_resp;
        int hf_resplen;
    };
};
```

The first two fields describe a buffer containing the command. The second two fields describe a buffer large enough to hold the expected response. Note that each command and response structure begins with a **virtual terminal data** (VTD) header. (See “General Output” on page 5-78 for an explanation of the VTD header.) The following query commands use this **ioctl** operation and can be found on the following pages:

- “Query Device IDs Command”
- “Query Physical Device Command” on page 5-59
- “Query Locator Command” on page 5-63
- “Query LPFKs Command” on page 5-64
- “Query Dials Command” on page 5-65
- “Query Presentation Space Command” on page 5-65
- “Query HFT Device Command” on page 5-66
- “Query DMA Command” on page 5-67

Query Device IDs Command

This command uses the **hfqdevidec** structure and contains the following fields and values:

Field	Value
hf-intro.hf-typehi	HFQDEVIDCH
hf-intro.hf-typelo	HFQDEVIDCL

This command fills the response buffer with the information about the display devices. The information is in the form of a **hfqdevidr** structure and contains the following fields and values:

Field	Value
hf-intro.hf-typehi	HFQDEVIDRH
hf-intro.hf-typelo	HFQDEVIDRL
hf-numdev	The number of devices for which data is reported.

The following fields are repeated for each physical device:

hf-devid	Physical device ID. The first device ID is the active display device ID, unless the change physical display command has changed the active display ID. The following values are possible:
-----------------	---

0x0401mmnn	IBM 5151 Display and 5151 Display and Printer Adapter
0x0402mmnn	IBM 6153 Display and 6153 Display Adapter
0x0403mmnn	IBM 5151 Display and 5154 Display Adapter
0x0404mmnn	IBM 5154 Display and 5154 Display Adapter
0x0405mmnn	IBM 6155 Display and 6155 Display Adapter
0x0406mmnn	IBM 6154 Display and 6154 Display Adapter
0x0408mmnn	IBM 5081 Display and 5081 Display Adapter

Note: The *mm* value indicates whether the adapter is totally functional. When this value is 0x00, the adapter is totally functional. Any other value indicates the adapter is less than fully functional or not working at all, but is present on the machine. The *nn* value can be from 0x01 to 0x04 and differentiates between multiple instances of the same adapter type.

hf-class	Display class (0x44).
-----------------	-----------------------

Query Physical Device Command

This command returns information about displays and locator devices. The **hfqphdevc** structure is used to issue this command and contains the following fields and values:

Field	Value
hf-intro.hf-typehi	HFQPDEVCH
hf-intro.hf-typelo	HFQPDEVCL
hf-phdevid	Physical device ID. The value 0 specifies the active device that is currently attached to the virtual terminal.

This command fills the response buffer with information about the displays and locator devices. This information is in the form of a **hfqphdevr** structure.

```
struct hfqphdevr
{
    char hf_intro[HFINTROSZ];
    char hf_sublen;
    char hf_subtype;
    /* locator device */
    char hf_scale[4];
    char hf_locattr[1];
    char hf_rsvd[3];
    /* display device */
    char hf_attr[4];
    char hf_pwidth[4];
    char hf_pheight[4];
    char hf_mwidth[4];
    char hf_mheight[4];
    char hf_bperpel[4];
    char hf_phdev[4];
    /* display font */
    char hf_numfont[4];
    /* remainder is of variable length */
    /* struct hffont hffont[N]; where N is value in hf_numfont */
    char hf_fontstart;
    /* following is one color response */
    /* struct hfcolor hfcolor; */
};

struct hfqfont
{
    char hf_fontid[4];
    char hf_fontstyle[4];
    char hf_fontattr[4];
    char hf_fontwidth[4];
    char hf_fontheight[4];
};

struct hfcolor
```

```
{
    char hf_numcolor[4];
    char hf_numactive[4];
    char hf_numfgrnd[4];
    char hf_numbgrnd[4];
    char hf_actcolor[4];
};
```

These structures are explained in the following sections:

- “Physical Device Information VTD Header”
- “Physical Locator Information”
- “Physical Display Device Information” on page 5-62
- “Physical Display Font Information” on page 5-62
- “Physical Display Color Information” on page 5-63.

Physical Device Information VTD Header

The information VTD header for the device contains the following fields and values:

Field	Value
hf_intro.hf_typehi	HFQPDEV RH
hf_intro.hf_typelo	HFQPDEV RL

Physical Locator Information

The locator device response information contains the following fields and values:

Field	Value
hf_scale	Scale factor (millimeters per 100 counts)
hf_locattr[0]	Locator attributes:
	HFLOCABS If set, the locator device reports absolute coordinates (for example, a tablet device). If not set, it reports relative coordinates (for example, a mouse).

Physical Display Device Information

The display device response information contains the following fields and values:

Field	Value
hf_attr[0]	Display device attributes: HFISAPA All-points-addressable (APA) display. HFHASBLINK Blink function allowed. All other values are reserved.
hf_attr[2]	Display device attributes: HFHACOLOR Color allowed. All other values are reserved.
hf_attr[3]	Display device attributes: HFCHGPALET Can change display adapter's color palette. All other values are reserved.
hf_pwidth	Displayable width of physical screen, expressed in pels or pixels for all displays.
hf_pheight	Displayable height of physical screen, expressed in pels for all displays.
hf_mwidth	Displayable width (in millimeters).
hf_mheight	Displayable height (in millimeters).
hf_bperpel	Bits per pel (1, 2, or 4).
hf_phdevi	Display device ID.

Physical Display Font Information

The display device font response information contains the following fields and values:

Field	Value
hf_numfont	Number of fonts available to this display. The following fields appear for each available font.
hf_fontid	Physical font ID.
hf_fontstyle	Physical font style. HFFNTVAR This font results in a variable presentation space depending on the display type used.

	HFFNTKSR	This font results in a 80 by 25 presentation space regardless of the display type used.
hf-fontattr[3]	Physical font attribute. This field can have the following values:	
	HFFNTPLAIN	Plain
	HFFNTBOLD	Bold
	HFFNTITALIC	Italic.
hf-fontwidth	Physical font width (the width of a character cell in pels).	
hf-fontheight	Physical font height (the height of a character cell in pels).	

Physical Display Color Information

The display device color response information contains the following fields and values:

Field	Value
hf-numcolor	Total number of colors possible
hf-numactive	Number of colors that can be active at any one time
hf-numfgrnd	Number of foreground color options
hf-numbgrnd	Number of background color options
hf-actcolor	Active color value. The value of this field can be in the range 0 to the total number of colors possible (hf-numcolor) minus 1. This field is repeated for each of the currently active colors.

Query Locator Command

This command returns information about the locator device. The **hfqgraphdev** structure is used to query the locators and contains the following fields and values:

Field	Value
hf-intro.hf-typehi	HFQLOCCH
hf-intro.hf-typelo	HFQLOCCL

This command fills the response buffer with the locator information. This information is in the form of a **hfqlocr** structure and contains the following fields and values:

Field	Value
hf-intro.hf-typehi	HFQLOCRH
hf-intro.hf-typelo	HFQLOCRL
hf-resolution	The resolution of the locator. The resolution is a 4-byte value expressed in millimeters per 100 count.

hf_devinfo[0]	Locator attributes:
	HFLOCABS If set, absolute coordinates (tablet). If not set, relative coordinates (mouse).
	HFLOCUNKNOWN Unknown sensor type, or the locator is a mouse.
	HFLOCSTYLUS The tablet has a stylus sensor.
	HFLOCPUCK The tablet has a puck sensor.
hf_horzmax_cnt	Horizontal maximum count (a 2-byte value).
hf_vertmax_cnt	Vertical maximum count (a 2-byte value).
hf_horzdead_zone	Horizontal tablet dead zone or mouse threshold.
hf_vertdead_zone	Vertical tablet dead zone or mouse threshold.

Query LPFKs Command

This command returns device information about the LPFKs. The **hfggraphdev** structure is used to query the LPFKs and contains the following fields and values:

Field	Value
hf_intro.hf_typehi	HFQLPFKSCH
hf_intro.hf_typelo	HFQLPFKSCL

This command fills the response buffer with LPFK information. This information is in the form of a **hfdial_lpfk** structure and contains the following fields and values:

Field	Value
hf_intro.hf_typehi	HFQLPFKSRH
hf_intro.hf_typelo	HFQLPFKSRL
hf_numlpfks	Number of LPFKs on the device.
hf_data2.lpfk.flags	A set of 32 bits corresponding to each of the LPFKs. Bits that are set to 1 indicate enabled LPFKs; bits set to 0 indicate disabled LPFKs.

Query Dials Command

This command returns device information about the dials. The **hfqgraphdev** structure is used to issue this command and contains the following fields and values:

Field	Value
hf-intro.hf-typehi	HFQDIALSCH
hf-intro.hf-typelo	HFQDIALSCL

This command fills the response buffer with dials information. This information is in the form of a **hfdial-lpfk** structure and contains the following fields and values:

Field	Value
hf-intro.hf-typehi	HFQDIALSRH
hf-intro.hf-typelo	HFQDIALSRL
hf-numdials	Number of dials on the device.
hf-data2.granularity	An array of 16 1-byte values giving the <i>granularity</i> of each dial. Granularity is the number of events per full 360° revolution of the dial. The values in the array represent powers of 2.

Query Presentation Space Command

This command returns an ASCII data steam image of the current display screen. Attribute and character set information on the queried block are returned. This query is valid only in KSR mode. (Note that this operation is called "Query ASCII Codes and Attributes" in *VRM Device Support*.)

The **hfqpresc** structure is used to issue this command and contains the following fields and values:

Field	Value
hf-intro.hf-typehi	HFQPRESCH
hf-intro.hf-typelo	HFQPRESCL
hf-sublen	2
hf-subtype	0
hf-xuleft	The upper-left X coordinate (first column of the block)
hf-yuleft	The upper-left Y coordinate (first row in the block)
hf-xlright	The lower-right X coordinate (last column in the block)
hf-ylright	The lower-right Y coordinate (last row in the block).

This command fills the response buffer with information about the attributes and character set. This information is in the form of a **hfqpresr** structure and contains the following fields and values:

Field	Value
hf-intro.hf-typehi	HFQPRESRH
hf-intro.hf-typelo	HFQPRESRL

The data returned from this command is an ASCII data stream that contains character codes from the queried block. Character set and attribute changes are indicated with SGR and SGO control sequences. A line feed control is returned after the last character code in each line of the queried block.

Note: The returned attributes can only be a subset of the original attributes that were output to the display. Only those attributes actually supported by the physical device are returned.

Query HFT Device Command

This command returns information about the **hft** device. The **hfqhftc** structure is used to issue this command and contains the following fields and values:

Field	Value
hf-intro.hf-typehi	HFQHFTCH
hf-intro.hf-typelo	HFQHFTCL

This command fills the response buffer with **hft** information. The **hfqhfttr** structure is used to issue this command and contains the following fields and values:

Field	Value
hf-intro.hf-typehi	HFQHFTTRH
hf-intro.hf-typelo	HFQHFTTRL
hf-phdevic	Physical display device ID (the same as returned by "Query Device IDs Command" on page 5-58)
hf-phrow	Number of character rows, based on the current font
hf-phcol	Number of character columns, based on the current font
hf-phcolor	Number of colors allowed on the display
hf-phfont	Number of fonts defined in the system
hf-phkbddid	Physical keyboard ID:
	HF101KBD 101-key keyboard
	HF102KBD 102-key keyboard
	HF106KBD 106-key keyboard.

Query DMA Command

This command returns the starting address and length of the application's DMA area. The **hfqdmac** structure is used to issue this command and contains the following fields and values:

Field	Value
hf-intro.hf-typehi	HFQDMACH
hf-intro.hf-typelo	HFQDMACL

This command returns the starting address and length of the application's DMA area. The **hfqhfr** structure is used to issue this command and contains the following fields and values:

Field	Value
hf-intro.hf-typehi	HFQDMARH
hf-intro.hf-typelo	HFQDMARL
hf-dmaaddr	Starting address of the DMA area
hf-dmalen	Length of the DMA area.

Enable Sound Signal (HFESOUND)

The HFESOUND operation informs the virtual terminal of the intent to use sound, enabling the routing of the sound response signal. The HFESOUND operation is invoked by the following:

```
int ioctl(fildes, HFESOUND, arg)
    int fildes;
    struct hfsmon *arg;

    struct hfsmon
    {
        int hf_momflags;
    };
};
```

The **hf_momflags** field contains one of the following values:

HFSINGLE	Only the process issuing the ioctl system call is to receive a sound response signal.
HFGROUP	All members of the current process group are to receive a sound response signal.

Disable Sound Signal (HFDSOUND)

THE HFDSOUND operation informs the virtual terminal of the intent to discontinue the use of sound. Sound response signals are not sent. The HFDSOUND operation is invoked by the following:

```
int ioctl(fildes, HFDSOUND, 0)
int fildes;
```

Enter Monitor Mode (HFSMON)

The HFSMON operation requests monitor mode. Monitor mode provides a program with direct control of the screen and keyboard. The HFSMON operation is invoked by the following:

```
int ioctl(fildes, HFSMON, arg)
int fildes;
struct hfsmon *arg;

struct hfsmon
{
    int hf_momflags;
};
```

The **hf_momflags** field contains one of the following values:

- HFSINGLE** Only the process issuing the **ioctl** system call is to receive monitor mode signals.
- HFGROUP** All members of the current process group are to receive monitor mode signals.

Exit Monitor Mode (HFCMON)

The HFCMON operation releases monitor mode. The HFCMON operation is invoked by the following:

```
int ioctl(fildes, HFCMON, 0)
int fildes;
```

Query Screen Manager (HFQSMGR)

The HFQSMGR operation queries the screen manager. The file descriptor must be associated with the screen manager `/dev/hft/mgr`. The HFQSMGR operation is invoked by the following:

```
int ioctl(fildes, HFQSMGR, arg)
    int fildes;
    struct hfbuf *arg;

    struct hfbuf
    {
        char *hf_bufp;
        int hf_buflen;
    };
};
```

The contents of the following **hfqstat** structure are stored in the memory area pointed to by **hf_bufp**.

```
struct hfqstat
{
    short hf_numvts;
    struct hfvtnfo
    {
        unsigned short hf_vtiodn;
        unsigned short hf_vtstate;
    } hf_vtinfo[HFNUMVTS];
};
```

The **hfqstat** structure contains the following fields and values:

Field	Description
-------	-------------

hf_numvts	The number of virtual terminals.
------------------	----------------------------------

The following fields are repeated for each virtual terminal:

hf_vtiodn	The virtual terminal IODN.
------------------	----------------------------

hf_vtstate	Status:
-------------------	---------

HFVTHIDDEN	The virtual terminal is hidden.
-------------------	---------------------------------

HFVTACTIVE	The virtual terminal is active.
-------------------	---------------------------------

HFVTCOMMAND	The virtual terminal is the command terminal.
--------------------	---

HFVTTRUSTED	The virtual terminal is a trusted terminal.
--------------------	---

Control Screen Manager (HFCSMGR)

The HFCSMGR operation requests the screen manager to manipulate the status of virtual terminals. Virtual terminals are linked together in a group called the **screen manager ring**. The screen manager places an entry in the ring for each virtual terminal opened. The terminal that is currently active is called the **head** of the ring; the last terminal on the ring is called the **tail**. When a new terminal is opened, that ring, the terminal becomes the head of the ring.

Two key sequences are used to switch between virtual terminals and control which terminal is currently active. The **active** terminal is the terminal that accepts data input devices. Pressing the **Alt-Action** key sequence makes the **next** virtual terminal active. This relationship is indicated by **a** in Figure 5-2. Pressing the **Shift-Action** key sequence on the active terminal makes the **last** virtual terminal active. This relationship is indicated by **b** in Figure 5-2.

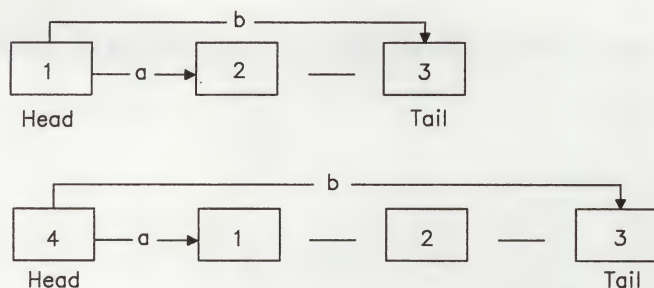


Figure 5-2. Screen Manager Ring Examples. In this figure, **a** indicates the path from the active virtual terminal to the next and **b** indicates the path from the active virtual terminal to the last.

Note that with three entries in the ring, all the terminals can be accessed with a single key sequence. With four or more entries, terminals can be skipped in some cases to activate a particular terminal. For example, in the preceding figure with four terminal entries, terminal #2 cannot be accessed from the keyboard of active terminal #4 without first skipping to terminal #1 or terminal #3.

The **hide** option of this command logically removes terminals from the ring. Hiding a terminal causes it to be bypassed when its position in the ring would ordinarily make it the active terminal.

The file descriptor must be associated with the screen manager `/dev/hft/mgr`. The HFCSMGR operation is invoked by the following:

```
int ioctl(fildes, HFCSMGR, arg)
```



```
int fildes;
struct hfsmgrcmd *arg;

struct hfsmgrcmd {
    int hf_cmd;
    int hf_vtid;
    int hf_vsid;
};
```

The **hf_vtid** and **hf_vsid** fields are set as follows:

hf_vtid The IODN of the virtual terminal

hf_vsid Reserved.

The **hf_cmd** field contains one of the following screen manager commands:

SMACT Activates the virtual terminal. This command places the virtual terminal specified by the IODN at the head of the screen manager ring, making it the active terminal. The terminal's hidden flag is also cleared. The screen manager cannot activate the virtual terminal if the currently active virtual terminal cannot be deactivated.

SMHIDE Hides the virtual terminal. This command marks the terminal identified by the IODN so that the screen manager will not activate it. This does not affect the terminal's position in the ring. When the hidden flag is set, the screen manager ignores the terminal's presence in the ring until an **SMUNHIDE** command is issued. If the virtual terminal is active when the hide command is issued, then the screen manager makes the terminal inactive (if possible), but does not prevent the virtual machine attached to it from communicating with it. Hiding the active virtual terminal has the same effect as the last window function. If all virtual terminals are hidden, then the physical display continues to show the contents of the last virtual terminal that was hidden.

SMSCMD Sets the command virtual terminal. This command designates a terminal as the command virtual terminal. The **command virtual terminal** is the terminal that is activated by pressing both locator buttons at the same time, or by pressing the **Ctrl-Action** key sequence.

SMUNHIDE Undoes the action performed by **SMHIDE**. The **hf_vtid** field contains the IODN of the virtual terminal where the command should be sent. The **hf_vsid** field is reserved.

This command restores the presence of the terminal in the ring, but does not affect its ring position or make it active. If the virtual terminal happens to be at the head of the ring when this command is issued, then it becomes visible and active.

- SMCVTEN** Causes the command virtual terminal to be activated when both locator buttons are pressed at the same time. This is the default setting. Since all virtual terminals are affected, programs that change this setting should restore it as soon as the locator is no longer needed.
- SMCVTDI** Causes input data to be reported when both locator buttons are pressed at the same time. The data reported is similar to that reported when a single button is pressed.

DMA Move (HFMDMA)

The HFDMA operation specifies the source, destination, and length of a monitor mode DMA move data operation. The HFDMA operation is invoked by the following:

```
int ioctl(fildev, HFMDMA, arg)
int fildev;
struct hfmdma *arg;

struct hfmdma
{
    uint hf_srcaddr; /* Virtual source address of DMA data */
    uint hf_dmalen; /* Length of data to be moved by DMA */
    uint hf_destaddr; /* Virtual destination address of DMA data */
}
```

This **ioctl** operation maps an application data area to segment E DMA space or unmaps an application data area. Mapping occurs when the source address specifies the application data area and the destination address specifies the segment E space. Unmapping takes place when the addresses are reversed from mapping. Pages are pinned when mapping takes place and unpinned during unmapping.

The range of space available for segment E DMA depends on the amount of memory installed in the system and on the amount of memory allocated to DMA space by the **HFCHNGDMA** option of the **HFRCONF** **ioctl** operation (see “Reconfigure (HFRCONF)” on page 5-48).

Warning: Once a data space is mapped onto segment E space with this **ioctl** operation, do not attempt to access the data space until after it is unmapped. Otherwise, loss of data, unpredictable results, and permanent depletion of system resources may occur.

Input

Data read from an **hft** device with the **read** system call can contain not only character data entered from a keyboard, but also input from other devices, such as a locator, a tablet, valuator, lighted programmable function keys, and displays. Data from devices other than the keyboard is passed back from the **read** system call in the form of special control sequences that are described in this section.

Note: These control sequences contain binary data. To prevent the binary data from being misinterpreted as ASCII control codes, set the terminal's canonical processing off. See "ICANON" on page 5-140 for details.

Untranslated Key Control

If keyboard input is received when HFXLATKBD is turned off, the following control sequence is returned. See "Protocol Modes" on page 5-79 for a description of HFXLATKBD. The key position identifies the physical key pressed. The key status bits indicate **Alt**, **Alt-Gr**, **Ctrl**, **Shift**, **Caps Lock**, and **Num Lock** key states. The scan code and make/break keys are dependent upon hardware and require knowledge of the physical keyboard in use. See "keyboard" on page 5-97 for additional information.

The structure of the untranslated key control is:

```
struct hfunxlate
{
    char hf_esc;
    char hf_lbr;
    char hf_ww;
    char hf_keypos;
    char hf_scancode;
    char hf_status[2];
};
```

The fields of the structure are:

Field	Description
hf_esc	ESC (0x1B)
hf_lbr	[(0x5B)
hf_ww	w (0x77)
hf_keypos	Key Position
hf_scancode	Scan Code (see "keyboard" on page 5-97)

hf_status[0]	Status:	
	HFUXSHIFT	A shift key is pressed.
	HFUXCTRL	Ctrl key is pressed.
	HFUXALT	Alt key is pressed.
	HFUXCAPS	Caps Lock mode is in effect.
	HFUXNUM	Num Lock mode is in effect.
	HFUXMAKE	If set, key has been pressed. If not set, key has been released.
hf_status[1]	Status:	
	HFUXRPT	Automatic repeat (typematic) state.
	HFUXLSH	Left shift state.
	HFUXRSH	Right shift state.
	HFUXLALT	Left alternate shift state.
	HFUXRALT	Right alternate shift state. (Alt-Gr for 102-key keyboards).

Input Device Report

This control reports input data from the mouse, tablet, LPFKs, or valuator dials. The data is reported in the form of an **hflocator** structure, and the following sections describe the fields of this structure for each type of input device.

Mouse Report

hf_esc	ESC (0x1B)
hf_lbr	[(0x5B)
hf_why	y (0x79)
hf_deltax	The X delta, a signed integer that holds the relative X delta accumulations in counts of 0.25 millimeters of the locator movement in two's-complement form. This information is sent to the virtual terminal to indicate horizontal movement since the last locator movement.
hf_deltay	The Y delta, a signed integer that holds the relative Y delta accumulations in counts of 0.25 millimeters of the locator movement in two's-complement form. This information is sent to the virtual terminal to indicate vertical movement since the last locator movement.
hf_seconds	Time of the locator report in whole seconds since system startup.
hf_sixtyths	The fractional part, of locator report time stamp, in a sixtieth of a second.

hf_buttons The status of the locator buttons. This information is sent to the virtual terminal to indicate a change in the status of the buttons since the last locator movement in the following manner:

HFBUTTON1 Button 1 has been pressed.
HFBUTTON2 Button 2 has been pressed.
HFBUTTON3 Button 3 has been pressed.

hf_stype 0

Tablet Report

hf_esc ESC (0x1B)

hf_lbr [(0x5B)

hf_why y (0x79)

hf_deltax The absolute X coordinate of the tablet sensor.

hf_deltay The absolute Y coordinate of the tablet sensor.

hf_seconds Time of the locator report in whole seconds since system startup.

hf_sixtyths The fractional part, of locator report time stamp, in a sixtieth of a second.

hf_buttons The status of the locator buttons. This information is sent to the virtual terminal to indicate a change in the status of the buttons since the last locator movement in the following manner:

HFBUTTON1 The left button has been pressed.
HFBUTTON2 The right button has been pressed.

hf_stype 1

LPFK Report

hf_esc ESC (0x1B)

hf_lbr [(0x5B)

hf_why y (0x79)

hf_deltax The LPFK number.

hf_deltay Reserved.

hf_seconds Time of the report in whole seconds since system startup.

hf_sixtyths The fractional part, of locator report time stamp, in a sixtieth of a second.

hf_buttons The status of the LPFK.

hf_stype 2

Valuator Dial Report

hf_esc ESC (0x1B)

hf_lbr [(0x5B)

hf_why y (0x79)

hf_deltax The dial number.

hf_deltay The dial value delta. This is a signed integer value in the dial's units of granularity (see "Set Dial Granularities" on page 5-83).

hf_seconds Time of the report in whole seconds since system startup.

hf_sixtyths The fractional part, of locator report time stamp, in a sixtieth of a second.

hf_buttons The status of the dial.

hf_stype 3

Adapter-Generated Input

Some adapters can return status information to MOM applications by way of a ring buffer. This status information is placed in the ring buffer with a VTA multibyte control (ESC [r). This feature is not available to KSR mode.

The information that immediately follows the control sequence includes a 1-byte queue ID and 20 bytes of data. Note that the hardware returns 16-bit words and that the bit-numbering conventions are reversed. See *IBM RT Hardware Technical Reference* for details on the data returned for each adapter status entry.

Status	QID	Data
FIFO mode entered	0x01	0x03 in first data byte, rest reserved.
PHIGS traversal started	0x01	0x05 in first data byte, rest reserved.
FIFO pick mode set	0x01	0x07 in first data byte, rest reserved.
CGA mode entered	0x01	0x09 in first data byte, rest reserved.
Traversal stopped	0x01	0x0B in first data byte, rest reserved.
Single-step mode completed	0x01	0x0F in first data byte, rest reserved.

Status	QID	Data
Echo cursor completed	0x01	0x11 in first data byte, rest reserved.
Defined pointer echo completed	0x01	0x13 in first data byte, rest reserved.
Remove cursor completed	0x01	0x15 in first data byte, rest reserved.
Clear frame buffer completed	0x01	0x17 in first data byte, rest reserved.
Load lookup table completed	0x01	0x21 in first data byte, rest reserved.
Set pick window size completed	0x01	0x27 in first data byte, rest reserved.
Reset FIFO pick mode completed	0x01	0x29 in first data byte, rest reserved.
Set blink mode completed	0x01	0x2D in first data byte, rest reserved.
Reset blink mode completed	0x01	0x2F in first data byte, rest reserved.
Initialization complete	0x02	0x01 in first data byte, rest reserved.
Traversal complete	0x03	No data, all 20 bytes reserved.
Pick occurred	0x04	Data words 1-5 set to reason extension words 1-10.
Buffer error FIFO overflow Illegal graphic order Illegal request code Invalid page Stack error Traversal error	0x05	Data words 1-5 set to reason extension words 1-10.
PELPRO task completed	0x06	No data, all 20 bytes reserved.
PELPRO pick	0x07	Data words 1-5 set to reason extension words 1-10.
PELPRO vertical sync.	0x08	No data, all 20 bytes reserved.
FIFO half full	0x09	No data, all 20 bytes reserved.
FIFO half empty	0x0A	No data, all 20 bytes reserved.
Synchronize	0x0B	Data words 1-5 set to reason extension codes 1-10.

General Output

ASCII data can be sent to the virtual terminal using the **write** system call along with data of any length. In addition, virtual terminal control structures can be sent to the virtual terminal using the **write** system call.

Each control structure is introduced by a *virtual terminal data* (VTD) character sequence. The VTD prefix consists of the ASCII codes **ESC**, **[**, and **x** (0x1B5B78). This prefix is followed by a length and an operation type code. The data that follows this structure depends on the type of control.

The **hfintro** structure looks like this:

```
{
    char hf_esc;
    char hf_lbr;
    char hf_ex;
    char hf_len[4];
    char hf_typehi;
    char hf_typelo;
};
```

The significant fields in the **hfintro** structure are:

- hf_len** The total number of bytes in the header and associated data, not including the 3-character VTD control sequence. In other words, the length is the total number of characters in the control sequence minus 3.
- hf_typehi** The high-order byte of the information type code.
- hf_typelo** The low-order byte of the information type code.

Note that **hf_typehi** and **hf_typelo** are called the major and minor data types in *VRM Device Support*. The values of **hf_typehi** and **hf_typelo** are documented with each command.

Because the **hfintro** structure is an odd number of bytes in length, it is designated as the character array **hf_intro[HFINTROSZ]** in the structures that define the various operation requests. This prevents the C compiler from inserting bytes into the structure to align the following fields on word boundaries. The **hf_typehi** and **hf_typelo** fields are referred to as **hf_intro.hf_typehi** and **hf_intro.hf_typelo** in this book.

All reserved and unused fields must be set to 0. You can set the entire structure to 0 and then fill in the appropriate fields.

Protocol Modes

Protocol modes determine how the virtual terminal will interpret, translate and return input data. Two bits control each mode. The first, in the **hf-select** field, indicates whether to use the current mode setting. If this bit is set, then the corresponding bit in **hf-value** indicates the new setting for the mode. The mode bits are set to the default value when the virtual terminal is opened. These defaults can be changed during configuration with the HFRCONF operation.

The **hfprotocol** structure gives the protocol definitions:

Field	Value
hf-intro.hf-typehi	HFKSRPROH or HFMOMPROH
hf-intro.hf-typelo	HFKSRPROL or HFMOMPROL
hf-sublen	2
hf-subtype	0
hf-select	Specifies which modes to change. A bit value of 1 specifies the mode represented by that bit to change.
hf-select[0]	Mode selectors: HFHOSTS HFXLATKBD
hf-select[1]	Mode selectors: HFWRAP HFLOCATOR HFLPFKS HFDIALS HFDINTR HFDINTRONLY HFJKANA
hf-value[0]	New mode values: HFHOSTS HFXLATKBD
hf-value[1]	New mode values: HFWRAP HFLOCATOR HFLPFKS HFDIALS HFDINTR HFDINTRONLY HFJKANA

When issuing this command, specify a type of either **HFKSRPROH**, **HFKSRPROL** or **HFMOMPROH**, **HFMOMPROL** depending on whether you are sending this command from within Keyboard Send-Receive mode (KSR) or monitor mode (MOM). Only certain protocol modes are valid in each of these modes, as shown in the following table. An attempt to set an invalid protocol mode results in rejection of the entire request.

Protocol Mode	When Valid	Meaning
HFHOSTS	KSR	<p>A 0 bit (default) means not to report shift key depressions. A 1 bit means report shift key depressions.</p> <p>HFHOSTS mode specifies whether to report keyboard status changes. If HFHOSTS mode is set, the keyboard status information is returned in the KSI ANSI control (see "Multibyte Controls" on page 4-13).</p>
HFXLATKBD	KSR, MOM	A 1 bit (default) specifies that the keyboard input is translated. A 0 bit indicates send key data as untranslated key controls. See "Untranslated Key Control" on page 5-73.
HFWRAP	KSR	A 1 bit (default) causes the cursor to wrap when the presentation space boundary is exceeded. A 0 bit specifies do not wrap the cursor.
HFLOCATOR	KSR, MOM	A 0 bit (default) disables the locator from sending data. A 1 bit enables the locator to send data.
HFLPFKS	KSR, MOM	A 0 bit (default) disables LPFK input. A 1 bit enables LPFK input.
HFDIALS	KSR, MOM	A 0 bit (default) disables dial (valuator) input. A 1 bit enables dial input.
HFDINTR	MOM	A 0 bit (default) indicates that display adapter status information is not to be sent to the host. A 1 bit specifies that the display status is to be sent.
HFDINTRONLY	MOM	A 0 bit (default) specifies not to restrict the use of the MOM input ring buffer. A 1 bit specifies to restrict the use of the MOM input ring buffer to display adapter status information only.

Protocol Mode	When Valid	Meaning
HFJKANA	KSR, MOM	A 0 bit (default) disables kana shift state. A 1 bit enables kana input (for use with Japanese licensed program only).

Set Keyboard LEDs

The structure for this command is **hfkled** and contains the following fields and values:

Field	Value
hf-intro.hf-typehi	HFKLEDCH
hf-intro.hf-typelo	HFKLEDCL
hf-sublen	2
hf-subtype	1
hf-ledselect	Indicates which of three LEDs to change: HFNUMLOCK The Num Lock LED HFCAPSLOCK The Caps Lock LED HFSCROLLOCK The Scroll Lock LED.
hf-ledvalue	Indicates the value to which to set the LEDs specified in hf-ledselect . LEDs that are specified with a 1 bit are set: HFNUMLOCK The Num Lock LED HFCAPSLOCK The Caps Lock LED HFSCROLLOCK The Scroll Lock LED.

Set Locator Thresholds

The locator device receives notice of horizontal and vertical movement. The delta of these movement events are monitored by the driver until the accumulated events exceed either the horizontal or vertical thresholds, or both. The locator device accumulates measurements at consecutive samplings. When a threshold is exceeded, the driver enqueues the information to the virtual terminal. When the status of the locator buttons change, the accumulated measurements are returned to the virtual terminal, even if these measurements do not exceed a threshold. The virtual terminal provides neither echoing nor positional management functions for the locator.

Each opened virtual terminal has its own threshold values. When a virtual terminal is opened, the threshold values default to 2.75 millimeters horizontal and 5.5 millimeters vertical. If the thresholds are 0, each event report is returned to the virtual terminal at the sampling rate supported by the locator device driver. See "Protocol Modes" on page 5-79 for information on how to receive reports.

The structure for this command is **hfloth** and contains the following fields and values:

Field	Value
hf-intro.hf-typehi	HFLOTHCH
hf-intro.hf-typelo	HFLOTHCL
hf-sublen	2
hf-subtype	1
hf-hthresh	Specifies the horizontal threshold in values from 0 to 32767 in units of 0.25 millimeters.
hf-vthresh	Specifies the vertical threshold in values from 0 to 32767 in units of 0.25 millimeters.

Set Tablet Dead Zones

Dead zones are areas of the tablet from which no input reports are generated. Each virtual terminal can set its own dead zones.

Initially, both of the dead zone values are set to 0, making the entire tablet active. Setting both values to 32767 completely disables tablet input, as does turning off **HFLOCATOR** in the protocol mode definition (see "Protocol Modes" on page 5-79).

The **hftdzone** structure is used for this command and contains the following fields and values:

Field	Value
hf-intro.hf-typehi	HFTDZCH
hf-intro.hf-typelo	HFTDZCL
hf-sublen	2
hf-subtype	1
hf-horizontal	A 2-byte non-negative value specified in units of 0.25 millimeters.
hf-vertical	A 2-byte non-negative value specified in units of 0.25 millimeters.

Set LPFKs

The **hfdial-lpfk** structure is used for this command, and it contains the following fields:

Field	Value
hf-intro.hf-typehi	HFLPFKSCH
hf-intro.hf-typelo	HFLPFKSCL
hf-sublen	2

hf_subtype	1
hf_mask.keys	A 4-byte bit mask numbered 0 to 31. Bits that are set specify LPFK flag values to change.
hf_data2.lpfk.flags	A 4-byte set of bits numbered 0 to 31. For LPFKs selected by hf_mask.keys , a 0 bit disables the LPFK, and a 1 bit enables the LPFK.

Set Dial Granularities

The **hfdial_lpfk** structure is used for this command, and it contains the following fields:

Field	Value
hf_intro.hf_typehi	HFDIALSCH
hf_intro.hf_typelo	HFDIALSCL
hf_sublen	2
hf_subtype	1
hf_mask.dials	A 4-byte bit mask numbered 0 to 31. Bits that are set specify dial granularity values to change.
hf_data2.granularity	An array of 16 1-byte values giving the <i>granularity</i> of each dial. Granularity is the number of events per full 360° revolution of the dial. The values in the array represent powers of 2, and they can range from 2 to 8.

Sound

This command sends output to the speaker. The mode byte determines whether to execute sound commands for the active virtual terminal and whether to interrupt the application after the sound command executes. No range check is made for the frequency or duration values.

The **hfsound** structure is used for this command and contains the following fields and values:

Field	Value
hf_intro.hf_typehi	HFSOUNDCH
hf_intro.hf_typelo	HFSOUNDCL
hf_sublen	2
hf_subtype	1

hf-mode

Mode:

HFSIGSOUND

If set, causes the **SIGSOUND** signal to be sent to the process when this sound command is executed or discarded. If not set, then no signal is sent.

HFEEXECALWAYS

If set, causes this sound command to be executed whether or not this virtual terminal is active. If not set, then the sound command is executed only if the terminal is active.

hf-dur

Duration in 1/128 seconds.

hf-freq

Frequency in hertz.

Cancel Sound

The cancel sound command removes all commands that do not want sound commands executed from the speaker device. Only the commands that have the **execute all sound to this terminal** flag are left in the active terminal queue. An inactive terminal ignores this command.

Sending a **cancel** and/or **enable sound** command flushes the speaker driver queue when a virtual terminal transition occurs. Regardless of whether the sound request is executed or purged, the virtual terminal receives a response if the response flag is set (bit 0 of sound command byte 0 is equal to 1).

The **hfcansnd** structure is used for this command and contains the following fields and values:

Field	Value
hf-intro.hf-typehi	HFCANSNDCH
hf-intro.hf-tyelo	HFCANSNDCL

Change Physical Display

This command changes the physical display to which the virtual terminal is logically attached.

The **hfchgdsp** structure is used for this command and contains the following fields and values:

Field	Value
hf-intro.hf-typehi	HFCHGDSPCH
hf-intro.hf-tyelo	HFCHGDSPCL
hf-sublen	2

hf_subtype	0
hf_mode	Bits 0-1 and 3-15 of these bytes are reserved. Bit 2 specifies the default or another value specified for the physical display. HFNONDEF If set, uses the identifier specified in bytes 10-13 for the physical display. If not set, no changes occur.
hf_rsvd1	Returned from the Query Display Device IDs command.
hf_devid	Physical display device identifier.
hf_rsvd2	If the requested ID is not in the configured list of available displays, no action is taken. If the requested ID is available, but a suitable font is not loaded, no action is taken. See "Change Fonts" on page 5-87 for a discussion of the results of a successful display change.

Note: If the physical terminal is changed, it may be necessary to change the **TERM** environment variable. See "TERM" on page 4-76 and "terminfo" on page 3-219.

Keyboard Send-Receive Mode (KSR) Output

A virtual terminal in KSR mode (default mode) has a presentation space (PS) of a fixed number of columns and lines. A symbol can be placed at any column, on any line, in the presentation space. Graphics from the KSR data stream are placed in the PS relative to the cursor position. Keyboard input also relates to the cursor position.

Two common modes for displaying graphics are *replace* and *insert*. In replace mode, a graphic character sent to a KSR terminal is placed at the cursor, replacing the symbol already there. In insert mode, a graphic character sent to a KSR terminal is also placed at the cursor, but the symbol at the cursor and all symbols on the same line are shifted right one column position on the line. Characters shifted from the last column on the line disappear.

Another mode determines cursor movement after the the last column position of a line. This mode, *automatic new line* (AUTONL), determines if the cursor wraps around to the first column position of the next line or stays at the last column on the current line.

If AUTONL is set, the cursor moves to the first column position of the following line. If the cursor happens to be on the bottom line of the presentation space, the presentation space scrolls up one line. If AUTONL is reset, the cursor stays on the last column of the current line.

Blank lines in the presentation space and erased character positions display in the active background color with normal attributes.

The following control sequences are valid only in a KSR-mode data stream.

Character Set Definition

Mapping of code point to display symbol can be altered. Each virtual terminal maintains character set mapping tables for two unique user-definable character sets called Unique One and Unique Two. These sets contain 256 ten-bit display symbol codes and are activated by the SG0 or SG1 control (see "Multibyte Controls" on page 4-13).

Note: Data is kept in display symbol form in the virtual terminal, and translation back to ASCII codes is done using the standard character set definitions.

The **hfcharset** structure is used for character set definition and contains the following fields and values:

Field	Value
hf-intro.hf-typehi	HFCHARSETH
hf-intro.hf-tyelo	HFCHARSETL
hf-sublen	2
hf-subtype	1

hf-setnum	User-defined character set
HFUNIQ1	Unique One (user-definable set 1)
HFUNIQ2	Unique Two (user-definable set 2)
hf-rsvd	Reserved
hf-code	10-bit display symbol code. This field may be repeated up to 256 times. See "display symbols" on page 4-26 for the values of the display symbols.

Set KSR Color Palette

This command specifies the color to associate with certain display adapters. The default color palettes are the ANSI 3.64 palette for character terminals and the PC color palette for all-points-addressable terminals. If the color specified is not supported by the adapter, the virtual display driver sets that color to the default for that mode.

The structure for this command is **hfcolorpal** and contains the following fields and values:

Field	Value
hf-intro.hf-typehi	HFCOLORPALH
hf-intro.hf-typelo	HFCOLORPALL
hf-sublen	2
hf-subtype	1
hf-numcolor	Number of entries in the palette.
hf-palet	Adapter-specific settings of the first entry in the color palette. These settings must be repeated for each entry of the color palette corresponding to the display adapter. See <i>IBM RT Hardware Technical Reference</i> for information about the display adapter.

Change Fonts

When a virtual terminal is first opened, and whenever it is changed, the assignment is made to the first font in the list of configured fonts. The virtual terminal initially tries to select a font that results in a presentation space of 80 columns by 25 rows. The first font with a normal appearance (not italics) that meets this criteria is chosen. If no fonts meet this criteria, the first font that can be displayed on the particular device is selected. All alternate fonts will be initialized to this chosen ID.

If the font is changed, the data currently in the presentation space is lost and the cursor reverts to the double underscore and is placed at the home position (first column, first row). If it is desirable to control fonts, the fonts should be explicitly set when opening a terminal or changing a display.

If the **change fonts** request is accepted and the installed fonts are a different size than the previous fonts, the presentation space size is adjusted to the number of rows and columns that fit on the physical display screen for the new font size.

See the `/usr/lib/samples/README.font` file for information about defining and selecting fonts.

The **hffont** structure is used for this request and contains the following fields and values:

Field	Value
hf_intro.hf_typehi	HFFONTH
hf_intro.hf_typelo	HFFONTL
hf_sublen	2
hf_subtype	1
hf_primary	Physical font ID of primary font attribute.
hf_alt1	Physical font ID of first alternate font attribute.
hf_alt2	Physical font ID of second alternate font attribute.
hf_alt3	Physical font ID of third alternate font attribute.
hf_alt4	Physical font ID of fourth alternate font attribute.
hf_alt5	Physical font ID of fifth alternate font attribute.
hf_alt6	Physical font ID of sixth alternate font attribute.
hf_alt7	Physical font ID of seventh alternate font attribute.

Cursor Representation

The cursor representation data format determines how the cursor is presented on the display screen.

The **hfcursor** structure is used for this request and contains the following fields and values:

Field	Value
hf_intro.hf_typehi	HFCURSORH
hf_intro.hf_typelo	HFCURSORL
hf_sublen	2
hf_subtype	0
hf_rsvd	Reserved.
hf_shape	Cursor shape: HFNONE No cursor. HFSINGLUS Single underscore. HFDBLUS Double underscore. HFHALFBLOB Lower half of illuminated character cell. HFMDLINE Double mid-character line. HFFULLBLOB Full illuminated character cell.

Monitor Mode (MOM) Output

Programs that operate the display in all-points-addressable mode should select the monitor mode of the virtual terminal. In this mode, the program performs output directly to the display adapter through a range of the memory mapped I/O bus, thus avoiding **write** system calls. Such a program can also read data from a circular buffer, thus avoiding **read** system calls. Some execution speed is gained by operating in this mode, but portability is sacrificed because the program depends on specific display adapters.

Notes:

1. Do not leave terminal open in monitor mode.
2. No more than one process should be open to a virtual terminal in monitor mode.

In order for a user program to switch from normal KSR mode to monitor mode, it must perform several mode changes using system calls. The display-sharing concept using virtual terminals causes the program in monitor mode to participate in the **next window** function by temporarily releasing the display. This is also accomplished using system calls. While the user program is active to the display, it performs output operations directly to the display hardware with memory mapped I/O ports.

Entering Monitor Mode

The first action the user program should perform is to request I/O bus access mode by opening the bus access pseudo device. See “bus” on page 5-19 for more information.

The next action that should be performed is to issue the **HFSMON ioctl** operation to enable monitor mode signals **SIGGRANT** and **SIGRETRACT** and specify the method used by processes to receive the signals. (See “Enter Monitor Mode (HFSMON)” on page 5-68.)

Next, the program should write a protocol mode control specifying the type **HFMOMPROH**, **HFMOMPROL**. See “Protocol Modes” on page 5-79.

After the successful completion of the three previous commands, the virtual terminal is in monitor mode.

Only a subset of ASCII codes are valid for the **write** system call while in monitor mode. All others are ignored.

Screen Request and Input Ring Buffer Definition

Although the virtual terminal is in monitor mode, the program can perform direct operations on the display hardware only when granted permission by the operating system. The program first writes a screen request control.

This request uses the **hfmomscreq** structure and contains the following fields and values:

Field	Value
hf_intro.hf_len	The length of the request up to and including the ring buffer.

hf-intro.hf-typehi	HFMOMREQH
hf-intro.hf-typelo	HFMOMREQL
hf-sublen	2
hf-subtype	0
hf-ringlen[2]	Shows the length of the ring in bytes.
hf-ringoffset[4]	Shows the offset to the input buffer ring (offset from the hf-ringlen field).

The **hf-ringlen** field specifies the size of the structure including the pointers and status fields. The program can directly access input key, locator, LPFK, and valuator data contained in the buffer without issuing **read** system calls.

The ring buffer structure **hfmomring** can be at any location in memory aligned on a word boundary. The **hf-ringoffset** is the difference between the ring buffer address and the address of **hf-ringlen** and must be a positive value. The **hfmomring** ring buffer structure is usually defined to immediately follow the **hfmomscreq** structure in memory. The compiler can implicitly insert one or more filler bytes between the two structures to align them at a memory address boundary. The value of **hf-ringoffset** must reflect such filler bytes. See the `/usr/lib/samples/hft/hftmom.c` source file for an example of how to calculate **hf-ringoffset**.

If you do not want to specify or use a ring buffer, then set the **hf-len** field of **hf-intro** to the size of the introducer only. In this case, read input with the standard **read** system call.

The structure of the **hfmomring** follows:

```
struct hfmomring
{
    char hf-rsvd[2];
    char hf-intreq;
    char hf-overflow;
    unsigned hf-source;
    unsigned hf-sink;
    int hf-unused[5];
    char hf-rdata[HFRDATA];
};
```


The **hfmomring** structure contains the following fields and values:

Field	Value
hf-rsvd	Reserved.
hf-intreq	Interrupt request can be set to 0xFF by the application to cause the virtual terminal subsystem to send a SIGMSG signal each time an input event occurs. If this flag is set to 0 (the default), then a signal is sent to the application only when the buffer goes from being empty to nonempty. This byte is automatically reset to 0 by the virtual terminal each time it stores input data into the ring buffer. See "Reading Input Data from the Ring Buffer" for further discussion.
hf-overflow	Overflow determines whether the input buffer ring can accommodate more input information. A value of 0xFF indicates an overflow. A value of 0x00 indicates normal operation.
hf-source	Ring offset for virtual terminal represents the offset into the input ring where the virtual terminal queues keyboard and locator input. This offset starts from the beginning of the ring, so the absolute minimum value for the virtual terminal offset is 32 bytes. Application programs must not alter this field. If a program attempts to alter it, then the virtual terminal is killed. See "Reading Input Data from the Ring Buffer" for further discussion.
hf-sink	Ring offset for application shows the offset into the input ring from which the application reads keyboard and locator information from the event queue. This offset also starts from the beginning of the input ring, so the minimum value for this offset is 32 bytes. See "Reading Input Data from the Ring Buffer" for further discussion.
hf-unused	Reserved.

Reading Input Data from the Ring Buffer

The program should start the offsets **hf-source** and **hf-sink** to be equal. This indicates a buffer empty condition. The program should then issue the **pause** system call, waiting for input. When the buffer goes from being empty to nonempty, the program receives a **SIGMSG** signal. Sending the **hfmomscreq** structure and defining the input ring buffer enables the sending of this signal. The program should extract characters from the ring buffer while incrementing the **hf-sink** offset for each character extracted, making sure to wrap around after reaching the end of the buffer. Care should be taken to ensure the buffer empty condition is properly detected. The program should test the equality of the offsets after it has updated the **hf-sink** offset. Therefore, the order of operation is: extract a character, update the offset in its memory location, and test the equality of offsets. If the offsets are equal, then set **hf-intreq** to 0xFF.

If **hf-source** == **hf-sink** - 1 (modulo ring size), then the ring buffer is full. If **hf-overflow** == 0xFF, then an overflow condition exists. The overflow condition indicates input data has been lost. The program resets the overflow condition by clearing **hf-overflow**.

Certain keys can be designated so they can be obtained using the **read** system call. This is particularly useful when such keys are the **Int** and **Quit** keys (see “termio” on page 5-133). These keys are designated using **HFSECHO**. Thus, by designating these keys in the break map, and by setting the **ISIG** mode of **termio**, it is possible to asynchronously interrupt a monitor mode program by pressing one of these keys.

Next Window Function

If a virtual terminal in monitor mode is active, pressing the Next Window key causes a **SIGRETRACT** signal to be sent to the process or group of processes specified by the **HFSMON ioctl** system call. Before activating the next virtual terminal, the operating system allows the program a chance to save the state of the display hardware, such as registers and refresh memory. After this is done, the program should write a screen release control to the terminal to inform the operating system that the state of the display hardware can be changed.

The screen release control is given by the **hfmomscrel** structure. This structure contains the following fields and values:

Field	Value
hf-intro.hf-len	The length of the entire structure, including the ring buffer, minus 3.
hf-intro.hf-typehi	HFMOMRELH
hf-intro.hf-typelo	HFMOMRELL

After the display is released, the next virtual terminal is activated. If this is not done within 30 seconds of the receipt of the **SIGRETRACT** signal, all processes in that terminal group receive a **SIGKILL** signal. This is a safeguard to prevent disabled programs from disrupting the next window function.

The program can issue a **pause** system call if there is no work to do while the display is not available. When the monitor mode virtual terminal is activated again with the Next Window key, the program receives a **SIGGRANT** signal. In other words, the program can resume direct output to the display. The display hardware state cannot be assumed to be the same as when the program released it.

Exiting Monitor Mode

When the program has no further use of the monitor mode, it should first write a screen release control, followed by a **KSR** protocol control. This is especially important if the virtual terminal is open by another process, such as the parent process, which is often the command shell. If the program is certain that no other processes have the terminal open, it can issue a **close** system call to remove that virtual terminal.

Next, a **HFCMON ioctl** operation should be issued to make sure that no monitor mode signals have been sent to this process or other processes in the terminal group in error.

Signals

In addition to the standard terminal signals (**SIGINT** and **SIGQUIT**), the virtual terminal generates other unique signals to inform the application program of asynchronous events. These signals include:

- | | |
|-------------------|---|
| SIGGRANT | Informs the user program that the display hardware can be directly accessed. This signal is sent following a monitor mode screen request VTD sequence. It is also sent after a monitor mode terminal is made active with the Next Window key. |
| SIGRETRACT | Informs the user program that the display hardware must be released for use by another program. This signal is sent after a monitor terminal is made inactive with the Next Window key. |
| SIGKILL | Sent to all processes in the terminal tty group to enforce the SIGRETRACT signal. If the user program does not respond with a screen release VTD sequence within 30 seconds after receiving a SIGRETRACT signal, the SIGKILL terminates all processes associated with that virtual terminal and the terminal is closed. |
| SIGMSG | Informs the user program that data has been placed into a previously empty input buffer. |

Considerations for hft Emulation

Communicating with an emulated or remote **hft** device presents a unique situation because the **ioctl** system call cannot be used. This is a result of the fact that **ioctl** passes data directly to the virtual terminal subsystem, bypassing the data stream. An **hft** emulator is usually connected through a pseudo-tty device, which means that all communication with it must be done through the data stream. Pseudo-tty devices are discussed under “pty” on page 5-126.

Therefore, two special multibyte control sequences can be used in place of invoking the **ioctl** system call, allowing applications to request an emulated **hft** to perform the **ioctl** operations. However, the **hft** device driver controlling the local console does not recognize these control sequences. The **ioctl** system call must be used to perform these operations on a **hft** device that is not emulated.

Both of these multibyte control sequences begin with a virtual terminal data (VTD) header. VTDs are explained under “General Output” on page 5-78.

To perform an **hft ioctl** operation whether or not the **hft** is emulated, an application should do the following:

1. Determine whether the **hft** device is being emulated. If the call **ioctl** (*fildes*, **IOCTYPE**, 0) returns the value **DD_PSEU**, *fildes* is a pseudo-tty device and can be connected to an **hft** emulator. Otherwise, the **hft** device is not emulated.
2. If the **hft** is not emulated, issue a regular **ioctl** system call, as outlined in “ioctl Operations” on page 5-46.
3. If the **hft** is emulated, do the following:
 - a. Set the pseudo-tty for raw data. That is, disable all input and output processing. This is necessary because the control sequences can contain binary data that would be misinterpreted by the pseudo-tty device driver as ASCII control codes. See “termio” on page 5-133 for details.
 - b. Use the **write** system call to send an **hfctlreq** VTD structure, immediately followed by the request structure, if any, that would normally be pointed to by the **ioctl arg** parameter. The **hfctlreq** structure contains the following fields and values:

Field	Value
hf-intro.hf-typehi	HFCTLREQH
hf-intro.hf-tyelo	HFCTLREQL
hf-request	The request type.
hf-arg-len	The length of the argument structure that follows the hfctlreq VTD, or 0 if none.

hf_rsp_len The maximum length of the response data structure that is to be returned with the **hfctlack** VTD. This value is 0 if no response buffer is expected.

- c. Using the **read** system call, read until an acknowledgement VTD is received. This acknowledgement takes the form of an **hfctlack** structure and is sometimes followed by a returned data structure, depending on the operation requested. The **hfctlack** structure contains the following fields and values:

Field	Value
hf_intro.hf_typehi	HFCTLACKH
hf_intro.hf_typelo	HFCTLACKL
hf_request	The request type.
hf_ret_code	The error code. A value of 0 indicates successful completion. A nonzero value is the value normally found in errno .
hf_arg_len	The length of the response data structure that follows the hfctlack VTD, or 0 if none. The length must not exceed the value of hf_rsp_len that was specified in the hfctlreq structure.

The file **/usr/lib/samples/hft/hftctl.c** contains a sample program that illustrates how to implement this procedure.

File

/dev/hft/*

Related Information

In this book: “**ioctl**” on page 2-407, “**fonts**” on page 3-79, “**data stream**” on page 4-5, “**config**” on page 5-21, and “**termio**” on page 5-133.

The discussion of the virtual terminal subsystem in *VRM Programming Support. Keyboard Description and Character Reference*.

keyboard

Purpose

Maps the 101-key RT keyboard.

Description

A keyboard mapping table is maintained for each virtual terminal. This table relates a key indicated by its key position along with the shift, control, or alternate keys to a character, mode processor function or string of characters. Portions or all of this mapping table can be modified by data passed to the **hfbuf** structure in the **HFSKBD ioctl** system call. See "hft" on page 5-39 for information about this **ioctl** system call. See *Keyboard Description and Character Reference* for details about other RT keyboards.

Each key on the standard RT keyboard has a numeric position code that is used for this field. Figure 5-3 matches the key to its position code.

110	112	113	114	115	116	117	118	119	120	121	122	123	124	125	126					
1	2	3	4	5	6	7	8	9	10	11	12	13	15	75	80	85	90	95	100	105
16	17	18	19	20	21	22	23	24	25	26	27	28	29	76	81	86	91	96	101	106
30	31	32	33	34	35	36	37	38	39	40	41	43					92	97	102	
44	46	47	48	49	50	51	52	53	54	55	57			83			93	98	103	108
58	60	61						62	64	79	84	89	99	104						

Figure 5-3. Position Codes for Remapping a Keyboard

keyboard

The following keys are not redefinable because their function is predefined at the VRM level.

Key Position	Function	States that cannot be remapped
30	Caps Lock key	All states
44	Left Shift key	All states
57	Right Shift key	All states
58	Ctrl key	All states
60	Left Alt key	All states
62	Right Alt key	All states
64	Action key	Shift, Control, Alternate, and Alternate Graphics states
90	Num Lock key	Base and Shift states

US 101-Key Keyboard Translate Table

The following table gives this information about the default mapping for the U.S. 101-key keyboard:

Key Posn – Keyboard key position.

Shift – The shift state of the position: Base, Shift, Ctrl, or Alt. (Note that the Alt Gr shift state is always the same as the Alt state.)

Assignment – The character or control assigned to that key.

Keyboard Definition – Provides information as it would appear as part of a keyboard definition structure. (See “Set Keyboard Map (HFSKBD)” on page 5-53 for details.) Within the table, interpret the fields as follows:

nnn – One-byte key position number being defined.

s – Shift state being defined:

b (base) – No **Shift** key is pressed.

s (shift) – Either **Left** or **Right Shift** key is pressed.

c (control) – **Ctrl** key is pressed.

a (alt) – **Alt** key is pressed.

t – Type of definition:

c – Regular character definition, followed by a 1-byte code page identifier and a 1-byte code point specification. The code page identifiers are:

> for Code Page P0

= for Code Page P1

> for Code Page P2

This identifier is followed by a 1-byte code point identifier, given in the table as a decimal number.

f – Function specification, followed by a 2-byte function identifier, which is specified in the table as a hexadecimal value.

s – String specification, followed by a 1-byte code page identifier, a 1-byte string length and the 1-byte code point identifiers within the specified code page. No string specifications are included in the default keyboard layouts.

d – Nonspacing or *dead* character definition, followed by a 1-byte code page identifier and a 1-byte code point specification. The code page identifiers are as described for character definition. This is followed by a 1-byte code point identifier, given in the table as a decimal number.

Returned String – Specifies the data that is returned to the program that is reading the keyboard.

Notes – Specifies additional information. Entries in this column include:

- **CL** – This key is affected by **Caps Lock**.
- **DK** – This key is a dead character on this key state.
- **P1** – This key is a character from Code Page P1.
- **P2** – This key is a character from Code Page P2.

The **Alt** key, followed by one or more numbered keys on the numeric pad, will return a single character which has the value entered on the numeric pad. The value accumulates while the **Alt** key is held down and returns when that key is released. Only spacing character codes and single-byte controls are produced by this method.

keyboard

Key Posn	Shift State	Assignment	Keyboard nnn s t	Returned String	Notes
1	Base	` Grave Accent	1 b c < 96	0x60	
1	Shift	~ Tilde Accent	1 s c < 126	0x7e	
1	Ctrl	PFK 57	1 c f 0x39	ESC [0 5 7 q	
1	Alt	PFK 115	1 a f 0x73	ESC [1 1 5 q	
2	Base	1 One	2 b c < 49	0x31	
2	Shift	! Exclamation Point	2 s c < 33	0x21	
2	Ctrl	PFK 49	2 c f 0x31	ESC [0 4 9 q	
2	Alt	PFK 58	2 a f 0x3a	ESC [0 5 8 q	
3	Base	2 Two	3 b c < 50	0x32	
3	Shift	@ At Sign	3 s c < 64	0x40	
3	Ctrl	NUL Null	3 c c < 0	0x00	
3	Alt	PFK 59	3 a f 0x3b	ESC [0 5 9 q	
4	Base	3 Three	4 b c < 51	0x33	
4	Shift	# Number Sign	4 s c < 35	0x23	
4	Ctrl	PFK 50	4 c f 0x32	ESC [0 5 0 q	
4	Alt	PFK 60	4 a f 0x3c	ESC [0 6 0 q	
5	Base	4 Four	5 b c < 52	0x34	
5	Shift	\$ Dollar Sign	5 s c < 36	0x24	
5	Ctrl	PFK 51	5 c f 0x33	ESC [0 5 1 q	
5	Alt	PFK 61	5 a f 0x3d	ESC [0 6 1 q	
6	Base	5 Five	6 b c < 53	0x35	
6	Shift	% Percent Sign	6 s c < 37	0x25	
6	Ctrl	PFK 52	6 c f 0x34	ESC [0 5 2 q	
6	Alt	PFK 62	6 a f 0x3e	ESC [0 6 2 q	
7	Base	6 Six	7 b c < 54	0x36	
7	Shift	^ Circumflex Accent	7 s c < 94	0x5e	
7	Ctrl	SS2 Single Shift 2	7 c c < 30	0x1e	
7	Alt	PFK 63	7 a f 0x3f	ESC [0 6 3 q	
8	Base	7 Seven	8 b c < 55	0x37	
8	Shift	& Ampersand	8 s c < 38	0x26	
8	Ctrl	PFK 53	8 c f 0x35	ESC [0 5 3 q	
8	Alt	PFK 64	8 a f 0x40	ESC [0 6 4 q	

keyboard

Key Posn	Shift State	Assignment		Keyboard Definition			Returned String	Notes
				nnn	s	t		
9	Base	8	Eight	9	b	c	< 56	0x38
9	Shift	*	Asterisk	9	s	c	< 42	0x2a
9	Ctrl	PFK	54	9	c	f	0x36	ESC [0 5 4 q
9	Alt	PFK	65	9	a	f	0x41	ESC [0 6 5 q
10	Base	9	Nine	10	b	c	< 57	0x39
10	Shift	(Left Parenthesis	10	s	c	< 40	0x28
10	Ctrl	PFK	55	10	c	f	0x37	ESC [0 5 5 q
10	Alt	PFK	66	10	a	f	0x42	ESC [0 6 6 q
11	Base	0	Zero	11	b	c	< 48	0x30
11	Shift)	Right Parenthesis	11	s	c	< 41	0x29
11	Ctrl	PFK	56	11	c	f	0x38	ESC [0 5 6 q
11	Alt	PFK	67	11	a	f	0x43	ESC [0 6 7 q
12	Base	-	Hyphen	12	b	c	< 45	0x2d
12	Shift	_	Underscore	12	s	c	< 95	0x5f
12	Ctrl	SS1	Single Shift 1	12	c	c	< 31	0x1f
12	Alt	PFK	68	12	a	f	0x44	ESC [0 6 8 q
13	Base	=	Equal Sign	13	b	c	< 61	0x3d
13	Shift	+	Plus Sign	13	s	c	< 43	0x2b
13	Ctrl	PFK	69	13	c	f	0x45	ESC [0 6 9 q
13	Alt	PFK	70	13	a	f	0x46	ESC [0 7 0 q
14	Not available on keyboard							
15	Base	BS	Back Space	15	b	c	< 8	0x08
15	Shift	BS	Back Space	15	s	c	< 8	0x08
15	Ctrl	DEL	Delete	15	c	c	< 127	0x7f
15	Alt	PFK	71	15	a	f	0x47	ESC [0 7 1 q
16	Base	HT	Horizontal Tab	16	b	c	< 9	0x09
16	Shift	CBT	Cursor Back Tab	16	s	f	0x105	ESC [Z
16	Ctrl	PFK	72	16	c	f	0x48	ESC [0 7 2 q
16	Alt	PFK	73	16	a	f	0x49	ESC [0 7 3 q
17	Base	q	Lowercase q	17	b	c	< 113	0x71
17	Shift	Q	Uppercase q	17	s	c	< 81	0x51
17	Ctrl	DC1	Device Control 1	17	c	c	< 17	0x11
17	Alt	PFK	74	17	a	f	0x4a	ESC [0 7 4 q

CL

keyboard

Key Posn	Shift State	Assignment	Keyboard Definition nnn s t	Returned String	Notes
18	Base	w Lowercase w	18 b c < 119	0x77	CL
18	Shift	W Uppercase w	18 s c < 87	0x57	
18	Ctrl	ETB End Trans Block	18 c c < 23	0x17	
18	Alt	PFK 75	18 a f 0x4b	ESC [0 7 5 q	
19	Base	e Lowercase e	19 b c < 101	0x65	CL
19	Shift	E Uppercase e	19 s c < 69	0x45	
19	Ctrl	ENQ Enquiry	19 c c < 5	0x05	
19	Alt	PFK 76	19 a f 0x4c	ESC [0 7 6 q	
20	Base	r Lowercase r	20 b c < 114	0x72	CL
20	Shift	R Uppercase r	20 s c < 82	0x52	
20	Ctrl	DC2 Device Control 2	20 c c < 18	0x12	
20	Alt	PFK 77	20 a f 0x4d	ESC [0 7 7 q	
21	Base	t Lowercase t	21 b c < 116	0x74	CL
21	Shift	T Uppercase t	21 s c < 84	0x54	
21	Ctrl	DC4 Device Control 4	21 c c < 20	0x14	
21	Alt	PFK 78	21 a f 0x4e	ESC [0 7 8 q	
22	Base	y Lowercase y	22 b c < 121	0x79	CL
22	Shift	Y Uppercase y	22 s c < 89	0x59	
22	Ctrl	EM End of Media	22 c c < 25	0x19	
22	Alt	PFK 79	22 a f 0x4f	ESC [0 7 9 q	
23	Base	u Lowercase u	23 b c < 117	0x75	CL
23	Shift	U Uppercase u	23 s c < 85	0x55	
23	Ctrl	NAK Not Acknowledge	23 c c < 21	0x15	
23	Alt	PFK 80	23 a f 0x50	ESC [0 8 0 q	
24	Base	i Lowercase i	24 b c < 105	0x69	CL
24	Shift	I Uppercase i	24 s c < 73	0x49	
24	Ctrl	HT Horizontal Tab	24 c c < 9	0x09	
24	Alt	PFK 81	24 a f 0x51	ESC [0 8 1 q	
25	Base	o Lowercase o	25 b c < 111	0x6f	CL
25	Shift	O Uppercase o	25 s c < 79	0x4f	
25	Ctrl	SI Shift In	25 c c < 15	0x0f	
25	Alt	PFK 82	25 a f 0x52	ESC [0 8 2 q	

keyboard

Key Posn	Shift State	Assignment		Keyboard Definition				Returned String	Notes
				nnn	s	t			
26	Base	p	Lowercase p	26	b	c	< 112	0x70	CL
26	Shift	P	Uppercase p	26	s	c	< 80	0x50	
26	Ctrl	DLE	Data Link Enabl	26	c	c	< 16	0x10	
26	Alt	PFK	83	26	a	f	0x53	ESC [0 8 3 q	
27	Base	[Left Bracket	27	b	c	< 91	0x5b	
27	Shift	{	Left Brace	27	s	c	< 123	0x7b	
27	Ctrl	ESC	Escape	27	c	c	< 27	0x1b	
27	Alt	PFK	84	27	a	f	0x54	ESC [0 8 4 q	
28	Base]	Right Bracket	28	b	c	< 93	0x5d	
28	Shift	}	Right Brace	28	s	c	< 125	0x7d	
28	Ctrl	SS3	Single Shift 3	28	c	c	< 29	0x1d	
28	Alt	PFK	85	28	a	f	0x55	ESC [0 8 5 q	
29	Base	\	Reverse Slash	29	b	c	< 92	0x5c	
29	Shift		Pipe Symbol	29	s	c	< 124	0x7c	
29	Ctrl	SS4	Single Shift 4	29	c	c	< 28	0x1c	
29	Alt	PFK	86	29	a	f	0x56	ESC [0 8 6 q	
30	Base	Caps Lock		None				Not Returned	
30	Shift	Caps Lock		None				Not Returned	
30	Ctrl	Caps Lock		None				Not Returned	
30	Alt	Caps Lock		None				Not Returned	
31	Base	a	Lowercase a	31	b	c	< 97	0x61	CL
31	Shift	A	Uppercase a	31	s	c	< 65	0x41	
31	Ctrl	SOH	Start of Header	31	c	c	< 1	0x01	
31	Alt	PFK	87	31	a	f	0x57	ESC [0 8 7 q	
32	Base	s	Lowercase s	32	b	c	< 115	0x73	CL
32	Shift	S	Uppercase s	32	s	c	< 83	0x53	
32	Ctrl	DC3	Device Control 3	32	c	c	< 19	0x13	
32	Alt	PFK	88	32	a	f	0x58	ESC [0 8 8 q	
33	Base	d	Lowercase d	33	b	c	< 100	0x64	CL
33	Shift	D	Uppercase d	33	s	c	< 68	0x44	
33	Ctrl	EOT	End of Transmission	33	c	c	< 4	0x04	
33	Alt	PFK	89	33	a	f	0x59	ESC [0 8 9 q	

keyboard

Key Posn	Shift State	Assignment	Keyboard Definition	Returned String	Notes
34	Base	f	Lowercase f	34 b c < 102 0x66	CL
34	Shift	F	Uppercase f	34 s c < 70 0x46	
34	Ctrl	ACK	Acknowledge	34 c c < 6 0x06	
34	Alt	PFK	90	34 a f 0x5a ESC [0 9 0 q	
35	Base	g	Lowercase g	35 b c < 103 0x67	CL
35	Shift	G	Uppercase g	35 s c < 71 0x47	
35	Ctrl	BEL	Bell	35 c c < 7 0x07	
35	Alt	PFK	91	35 a f 0x5b ESC [0 9 1 q	
36	Base	h	Lowercase h	36 b c < 104 0x68	CL
36	Shift	H	Uppercase h	36 s c < 72 0x48	
36	Ctrl	BS	Backspace	36 c c < 8 0x08	
36	Alt	PFK	92	36 a f 0x5c ESC [0 9 2 q	
37	Base	j	Lowercase j	37 b c < 106 0x6a	CL
37	Shift	J	Uppercase j	37 s c < 74 0x4a	
37	Ctrl	LF	Line Feed	37 c c < 10 0x0a	
37	Alt	PFK	93	37 a f 0x5d ESC [0 9 3 q	
38	Base	k	Lowercase k	38 b c < 107 0x6b	CL
38	Shift	K	Uppercase k	38 s c < 75 0x4b	
38	Ctrl	VT	Vertical Tab	38 c c < 11 0x0b	
38	Alt	PFK	94	38 a f 0x5e ESC [0 9 4 q	
39	Base	l	Lowercase l	39 b c < 108 0x6c	CL
39	Shift	L	Uppercase l	39 s c < 76 0x4c	
39	Ctrl	FF	Form Feed	39 c c < 12 0x0c	
39	Alt	PFK	95	39 a f 0x5f ESC [0 9 5 q	
40	Base	;	Semicolon	40 b c < 59 0x3b	
40	Shift	:	Colon	40 s c < 58 0x3a	
40	Ctrl	PFK	96	40 c f 0x60 ESC [0 9 6 q	
40	Alt	PFK	97	40 a f 0x61 ESC [0 9 7 q	
41	Base	'	Quote, Apostrophe	41 b c < 39 0x27	
41	Shift	"	Double Quote	41 s c < 34 0x22	
41	Ctrl	PFK	98	41 c f 0x62 ESC [0 9 8 q	
41	Alt	PFK	99	41 a f 0x63 ESC [0 9 9 q	

keyboard

Key Posn	Shift State	Assignment		Keyboard Definition				Returned String	Notes
				nnn	s	t			
42	Not available on keyboard								
43	Base	CR	Carriage Return	43	b	c	< 13	0x0d	
43	Shift	CR	Carriage Return	43	s	c	< 13	0x0d	
43	Ctrl	CR	Carriage Return	43	c	c	< 13	0x0d	
43	Alt	PFK	100	43	a	f	0x64	ESC [1 0 0 q	
44	Base	Shift (Left)		None				Not Returned	
44	Shift	Shift (Left)		None				Not Returned	
44	Ctrl	Shift (Left)		None				Not Returned	
44	Alt	Shift (Left)		None				Reserved for IBM 5080	
45	Not available on keyboard								
46	Base	z	Lowercase z	46	b	c	< 122	0x7a	CL
46	Shift	Z	Uppercase z	46	s	c	< 90	0x5a	
46	Ctrl	SUB	Substitute Char.	46	c	c	< 26	0x1a	
46	Alt	PFK	101	46	a	f	0x65	ESC [1 0 1 q	
47	Base	x	Lowercase x	47	b	c	< 120	0x78	CL
47	Shift	X	Uppercase x	47	s	c	< 88	0x58	
47	Ctrl	CAN	Cancel	47	c	c	< 24	0x18	
47	Alt	PFK	102	47	a	f	0x66	ESC [1 0 2 q	
48	Base	c	Lowercase c	48	b	c	< 99	0x63	CL
48	Shift	C	Uppercase c	48	s	c	< 67	0x43	
48	Ctrl	ETX	End of Text	48	c	c	< 3	0x03	
48	Alt	PFK	103	48	a	f	0x67	ESC [1 0 3 q	
49	Base	v	Lowercase v	49	b	c	< 118	0x76	CL
49	Shift	V	Uppercase v	49	s	c	< 86	0x56	
49	Ctrl	SYN	Synch Idle	49	c	c	< 22	0x16	
49	Alt	PFK	104	49	a	f	0x68	ESC [1 0 4 q	
50	Base	b	Lowercase b	50	b	c	< 98	0x62	CL
50	Shift	B	Uppercase b	50	s	c	< 66	0x42	
50	Ctrl	STX	Start of Text	50	c	c	< 2	0x02	
50	Alt	PFK	105	50	a	f	0x69	ESC [1 0 5 q	
51	Base	n	Lowercase n	51	b	c	< 110	0x6e	CL
51	Shift	N	Uppercase n	51	s	c	< 78	0x4e	

keyboard

Key Posn	Shift State	Assignment		Keyboard Definition			Returned String	Notes	
				nnn	s	t			
51	Ctrl	SO	Shift Out	51	c	c	< 14	0x0e	
51	Alt	PFK	106	51	a	f	0x65	ESC [1 0 6 q	
52	Base	m	Lowercase m	52	b	c	< 109	0x6d	CL
52	Shift	M	Uppercase m	52	s	c	< 77	0x4d	
52	Ctrl	CR	Carriage Return	52	c	c	< 13	0x0d	
52	Alt	PFK	107	52	a	f	0x66	ESC [1 0 7 q	
53	Base	,	Comma	53	b	c	< 44	0x2c	
53	Shift	<	Less Than Sign	53	s	c	< 60	0x3c	
53	Ctrl	PFK	108	53	c	f	0x6c	ESC [1 0 8 q	
53	Alt	PFK	109	53	a	f	0x6d	ESC [1 0 9 q	
54	Base	.	Period	54	b	c	< 46	0x2e	
54	Shift	>	Greater Than Sign	54	s	c	< 62	0x3e	
54	Ctrl	PFK	110	54	c	f	0x6e	ESC [1 1 0 q	
54	Alt	PFK	111	54	a	f	0x6f	ESC [1 1 1 q	
55	Base	/	Slash	55	b	c	< 47	0x2f	
55	Shift	?	Question Mark	55	s	c	< 63	0x3f	
55	Ctrl	PFK	112	55	c	f	0x70	ESC [1 1 2 q	
55	Alt	PFK	113	55	a	f	0x71	ESC [1 1 3 q	
56	Not available on keyboard								
57	Base	Shift (Right)		None			Not Returned		
57	Shift	Shift (Right)		None			Not Returned		
57	Ctrl	Shift (Right)		None			Not Returned		
57	Alt	Shift (Right)		None			Reserved for IBM 5080		
58	Base	Control		None			Not Returned		
58	Shift	Control		None			Not Returned		
58	Ctrl	Control		None			Not Returned		
58	Alt	Control		None			Not Returned		
59	Not available on keyboard								
60	Base	Alternate Shift		None			Not Returned		
60	Shift	Alternate Shift		None			Not Returned		
60	Ctrl	Alternate Shift		None			Not Returned		
60	Alt	Alternate Shift		None			Not Returned		

keyboard

Key Posn	Shift State	Assignment		Keyboard Definition				Returned String	Notes
				nnn	s	t			
61	Base	SP	Space	61	b	c	< 32	0x20	
61	Shift	SP	Space	61	s	c	< 32	0x20	
61	Ctrl	SP	Space	61	c	c	< 32	0x20	
61	Alt	SP	Space	61	a	c	< 32	0x20	
62	Base	Alternate Graphic Shift		None				Not Returned	
62	Shift	Alternate Graphic Shift		None				Not Returned	
62	Ctrl	Alternate Graphic Shift		None				Not Returned	
62	Alt	Alternate Graphic Shift		None				Not Returned	
63	Not available on keyboard								
64	Base	PFK	114	64	b	f	0x72	ESC [1 1 4 q	
64	Shift	VTRM Previous Window		None				VTRM Previous Window	
64	Ctrl	VTRM Windows Window		None				VTRM Windows Window	
64	Alt	VTRM Next Window		None				VTRM Next Window	
65 - 74	Not available on keyboard								
75	Base	PFK	139 INS Toggle	75	b	f	0x8b	ESC [1 3 9 q	
75	Shift	PFK	139 INS Toggle	75	s	f	0x8b	ESC [1 3 9 q	
75	Ctrl	PFK	140	75	c	f	0x8c	ESC [1 4 0 q	
75	Alt	PFK	141	75	a	f	0x8d	ESC [1 4 1 q	
76	Base	DCH	Delate Character	76	b	f	0x151	ESC [P	
76	Shift	DCH	Delate Character	76	s	f	0x151	ESC [P	
76	Ctrl	PFK	142	76	c	f	0x8e	ESC [1 4 2 q	
76	Alt	DL	Delete Line	76	a	f	0x153	ESC [M	
77	Not available on keyboard								
78	Not available on keyboard								
79	Base	CUB	Cursor Back	79	b	f	0x104	ESC [D	
79	Shift	PFK	158	79	s	f	0x9e	ESC [1 5 8 q	
79	Ctrl	PFK	159	79	c	f	0x9f	ESC [1 5 9 q	
79	Alt	PFK	160	79	a	f	0xa0	ESC [1 6 0 q	
80	Base	HOME		80	b	f	0x108	ESC [H	
80	Shift	PFK	143	80	s	f	0x8f	ESC [1 4 3 q	
80	Ctrl	PFK	144	80	c	f	0x90	ESC [1 4 4 q	

keyboard

Key Posn	Shift State	Assignment		Keyboard Definition				Returned String	Notes
nnn	s	t							
80	Alt	PFK	145	80	a	f	0x91	ESC [1 4 5 q	
81	Base	PFK	146	81	b	f	0x92	ESC [1 4 6 q	
81	Shift	PFK	147	81	s	f	0x93	ESC [1 4 7 q	
81	Ctrl	PFK	148	81	c	f	0x94	ESC [1 4 8 q	
81	Alt	PFK	149	81	a	f	0x95	ESC [1 4 9 q	
82	Not available on keyboard								
83	Base	CUU	Cursor Up	83	b	f	0x101	ESC [A	
83	Shift	PFK	161	83	s	f	0xa1	ESC [1 6 1 q	
83	Ctrl	PFK	162	83	c	f	0xa2	ESC [1 6 2 q	
83	Alt	PFK	163	83	a	f	0xa3	ESC [1 6 3 q	
84	Base	CUD	Cursor Down	84	b	f	0x102	ESC [B	
84	Shift	PFK	164	84	s	f	0xa4	ESC [1 6 4 q	
84	Ctrl	PFK	165	84	c	f	0xa5	ESC [1 6 5 q	
84	Alt	PFK	166	84	a	f	0xa6	ESC [1 6 6 q	
85	Base	PFK	150	85	b	f	0x96	ESC [1 5 0 q	
85	Shift	PFK	151	85	s	f	0x97	ESC [1 5 1 q	
85	Ctrl	PFK	152	85	c	f	0x98	ESC [1 5 2 q	
85	Alt	PFK	153	85	a	f	0x99	ESC [1 5 3 q	
86	Base	PFK	154	86	b	f	0x9a	ESC [1 5 4 q	
86	Shift	PFK	155	86	s	f	0x9b	ESC [1 5 5 q	
86	Ctrl	PFK	156	86	c	f	0x9c	ESC [1 5 6 q	
86	Alt	PFK	157	86	a	f	0x9d	ESC [1 5 7 q	
87	Not available on keyboard								
88	Not available on keyboard								
89	Base	CUF	Cursor Forward	89	b	f	0x103	ESC [C	
89	Shift	PFK	167	89	s	f	0xa7	ESC [1 6 7 q	
89	Ctrl	PFK	168	89	c	f	0xa8	ESC [1 6 8 q	
89	Alt	PFK	169	89	a	f	0xa9	ESC [1 6 9 q	
90	Base	Num Lock		None				Not Returned	
90	Shift	Num Lock		None				Not Returned	
90	Ctrl	DC3	Device Control 3	90	c	c	< 19	0x13	

keyboard

Key Posn	Shift State	Assignment	Keyboard nnn s t Definition	Returned String	Notes
90	Alt	PFK 170	90 a f 0xaa	ESC [1 7 0 q	
91	Base	┐ Upper Left Corner	91 b c < 218	0xda	
91	Shift	7 Seven	91 s c < 55	0x37	
91	Ctrl	PFK 172	91 c f 0xac	ESC [1 7 2 q	
91	Alt	Alt+Num Entry	None	Return at Alt Break	
92	Base	└ Left Edge Int.	92 b c < 195	0xc3	
92	Shift	4 Four	92 s c < 52	0x34	
92	Ctrl	PFK 174	92 c f 0xae	ESC [1 7 4 q	
92	Alt	Alt+Num Entry	None	Return at Alt Break	
93	Base	└ Lower Left Corner	93 b c < 192	0xc0	
93	Shift	1 One	93 s c < 49	0x31	
93	Ctrl	PFK 176	93 c f 0xb0	ESC [1 7 6 q	
93	Alt	Alt+Num Entry	None	Return at Alt Break	
94	Not available on keyboard				
95	Base	/ Slash	95 b c < 47	0x2f	
95	Shift	/ Slash	95 s c < 47	0x2f	
95	Ctrl	PFK 179	95 c f 0xb3	ESC [1 7 9 q	
95	Alt	PFK 180	95 a f 0xb4	ESC [1 8 0 q	
96	Base	┘ Top Intersection	96 b c < 194	0xc2	
96	Shift	8 Eight	96 s c < 56	0x38	
96	Ctrl	PFK 182	96 c f 0xb6	ESC [1 8 2 q	
96	Alt	Alt+Num Entry	None	Return at Alt Break	
97	Base	┘ Center Intersection	97 b c < 197	0xc5	
97	Shift	5 Five	97 s c < 53	0x35	
97	Ctrl	PFK 184	97 c f 0xb8	ESC [1 8 4 q	
97	Alt	Alt+Num Entry	None	Return at Alt Break	
98	Base	└ Bottom Junction	98 b c < 193	0xc1	
98	Shift	2 Two	98 s c < 50	0x32	
98	Ctrl	PFK 186	98 c f 0xba	ESC [1 8 6 q	
98	Alt	Alt+Num Entry	None	Return at Alt Break	
99	Base	Vertical Bar	99 b c < 179	0xb3	
99	Shift	0 Zero	99 s c < 48	0x30	

keyboard

Key Posn	Shift State	Assignment	Keyboard Definition nnn s t	Returned String	Notes
99	Ctrl	PFK 178	99 c f 0xb2	ESC [1 7 8 q	
99	Alt	Alt+Num Entry	None	Return at Alt Break	
100	Base	* Asterisk	100 b c < 42	0x2a	
100	Shift	* Asterisk	100 s c < 42	0x2a	
100	Ctrl	PFK 187	100 c f 0xbb	ESC [1 8 7 q	
100	Alt	PFK 188	100 a f 0xbc	ESC [1 8 8 q	
101	Base	7 Upper Right Corner	101 b c < 191	0xbf	
101	Shift	9 Nine	101 s c < 57	0x39	
101	Ctrl	PFK 190	101 c f 0xbe	ESC [1 9 0 q	
101	Alt	Alt+Num Entry	None	Return at Alt Break	
102	Base	4 Right Edge Int.	102 b c < 180	0xb4	
102	Shift	6 Six	102 s c < 54	0x36	
102	Ctrl	PFK 192	102 c f 0xc0	ESC [1 9 2 q	
102	Alt	Alt+Num Entry	None	Return at Alt Break	
103	Base	J Lower Right Corner	103 b c < 217	0xd9	
103	Shift	3 Three	103 s c < 51	0x33	
103	Ctrl	PFK 194	103 c f 0xc2	ESC [1 9 4 q	
103	Alt	Alt+Num Entry	None	Return at Alt Break	
104	Base	- Horizontal Line	104 b c < 196	0xc4	
104	Shift	. Period	104 s c < 46	0x2e	
104	Ctrl	PFK 196	104 c f 0xc4	ESC [1 9 6 q	
104	Alt	PFK 197	104 a f 0xc5	ESC [1 9 7 q	
105	Base	- Hyphen, Minus Sign	105 b c < 45	0x2d	
105	Shift	- Hyphen, Minus Sign	105 s c < 45	0x2d	
105	Ctrl	PFK 198	105 c f 0xc6	ESC [1 9 8 q	
105	Alt	PFK 199	105 a f 0xc7	ESC [1 9 9 q	
106	Base	+ Plus Sign	106 b c < 43	0x2b	
106	Shift	+ Plus Sign	106 s c < 43	0x2b	
106	Ctrl	PFK 200	106 c f 0xc8	ESC [2 0 0 q	
106	Alt	PFK 201	106 a f 0xc9	ESC [2 0 1 q	
107	Not available on keyboard				

keyboard

Key Posn	Shift State	Assignment		Keyboard Definition			Returned String	Notes
				nnn	s	t		
108	Base	CR	Carriage Return	108	b	c	< 13	0x0d
108	Shift	CR	Carriage Return	108	s	c	< 13	0x0d
108	Ctrl	CR	Carriage Return	108	c	c	< 13	0x0d
108	Alt	PFK	100	108	a	f	0x64	ESC [1 0 0 q
109	Not available on keyboard							
110	Base	ESC	Escape	110	b	c	< 27	0x1b
110	Shift	PFK	120	110	s	f	0x78	ESC [1 2 0 q
110	Ctrl	PFK	121	110	c	f	0x79	ESC [1 2 1 q
110	Alt	PFK	122	110	a	f	0x7a	ESC [1 2 2 q
111	Not available on keyboard							
112	Base	PFK	1	112	b	f	0x01	ESC [0 0 1 q
112	Shift	PFK	13	112	s	f	0x0d	ESC [0 1 3 q
112	Ctrl	PFK	25	112	c	f	0x19	ESC [0 2 5 q
112	Alt	PFK	37	112	a	f	0x25	ESC [0 3 7 q
113	Base	PFK	2	113	b	f	0x02	ESC [0 0 2 q
113	Shift	PFK	14	113	s	f	0x0e	ESC [0 1 4 q
113	Ctrl	PFK	26	113	c	f	0x1a	ESC [0 2 6 q
113	Alt	PFK	38	113	a	f	0x26	ESC [0 3 8 q
114	Base	PFK	3	114	b	f	0x03	ESC [0 0 3 q
114	Shift	PFK	15	114	s	f	0x0f	ESC [0 1 5 q
114	Ctrl	PFK	27	114	c	f	0x1b	ESC [0 2 7 q
114	Alt	PFK	39	114	a	f	0x27	ESC [0 3 9 q
115	Base	PFK	4	115	b	f	0x04	ESC [0 0 4 q
115	Shift	PFK	16	115	s	f	0x10	ESC [0 1 6 q
115	Ctrl	PFK	28	115	c	f	0x1c	ESC [0 2 8 q
115	Alt	PFK	40	115	a	f	0x28	ESC [0 4 0 q
116	Base	PFK	5	116	b	f	0x05	ESC [0 0 5 q
116	Shift	PFK	17	116	s	f	0x11	ESC [0 1 7 q
116	Ctrl	PFK	29	116	c	f	0x1d	ESC [0 2 9 q
116	Alt	PFK	41	116	a	f	0x29	ESC [0 4 1 q
117	Base	PFK	6	117	b	f	0x06	ESC [0 0 6 q
117	Shift	PFK	18	117	s	f	0x12	ESC [0 1 8 q

keyboard

Key Posn	Shift State	Assignment	Keyboard Definition	Returned String	Notes
117	Ctrl	PFK 30	117 c f 0x1e	ESC [0 3 0 q	
117	Alt	PFK 42	117 a f 0x2a	ESC [0 4 2 q	
118	Base	PFK 7	118 b f 0x07	ESC [0 0 7 q	
118	Shift	PFK 19	118 s f 0x13	ESC [0 1 9 q	
118	Ctrl	PFK 31	118 c f 0x1f	ESC [0 3 1 q	
118	Alt	PFK 43	118 a f 0x2b	ESC [0 4 3 q	
119	Base	PFK 8	119 b f 0x08	ESC [0 0 8 q	
119	Shift	PFK 20	119 s f 0x14	ESC [0 2 0 q	
119	Ctrl	PFK 32	119 c f 0x20	ESC [0 3 2 q	
119	Alt	PFK 44	119 a f 0x2c	ESC [0 4 4 q	
120	Base	PFK 9	120 b f 0x09	ESC [0 0 9 q	
120	Shift	PFK 21	120 s f 0x15	ESC [0 2 1 q	
120	Ctrl	PFK 33	120 c f 0x21	ESC [0 3 3 q	
120	Alt	PFK 45	120 a f 0x2d	ESC [0 4 5 q	
121	Base	PFK 10	121 b f 0x0a	ESC [0 1 0 q	
121	Shift	PFK 22	121 s f 0x16	ESC [0 2 2 q	
121	Ctrl	PFK 34	121 c f 0x22	ESC [0 3 4 q	
121	Alt	PFK 46	121 a f 0x2e	ESC [0 4 6 q	
122	Base	PFK 11	122 b f 0x0b	ESC [0 1 1 q	
122	Shift	PFK 23	122 s f 0x17	ESC [0 2 3 q	
122	Ctrl	PFK 35	122 c f 0x23	ESC [0 3 5 q	
122	Alt	PFK 47	122 a f 0x2f	ESC [0 4 7 q	
123	Base	PFK 12	123 b f 0x0c	ESC [0 1 2 q	
123	Shift	PFK 24	123 s f 0x18	ESC [0 2 4 q	
123	Ctrl	PFK 36	123 c f 0x24	ESC [0 3 6 q	
123	Alt	PFK 48	123 a f 0x30	ESC [0 4 8 q	
124	Base	PFK 209	124 b f 0xd1	ESC [2 0 9 q	
124	Shift	PFK 210	124 s f 0xd2	ESC [2 1 0 q	
124	Ctrl	PFK 211	124 c f 0xd3	ESC [2 1 1 q	
124	Alt	PFK 212	124 a f 0xd4	ESC [2 1 2 q	
125	Base	PFK 213	125 b f 0xd5	ESC [2 1 3 q	
125	Shift	PFK 214	125 s f 0xd6	ESC [2 1 4 q	
125	Ctrl	PFK 215	125 c f 0xd7	ESC [2 1 5 q	

keyboard

Key Posn	Shift State	Assignment	Keyboard Definition nnn s t	Returned String	Notes
125	Alt	PFK 216	125 a f 0xd8	ESC [2 1 6 q	
126	Base	PFK 217	126 b f 0xd9	ESC [2 1 7 q	
126	Shift	PFK 218	126 s f 0xda	ESC [2 1 8 q	
126	Ctrl	DEL	126 c c < 127	0x7f	
126	Alt	DEL	126 a c < 127	0x7f	

Keystroke Control Sequences for System Functions

The following keystroke combinations cause the indicated system functions to be performed. The notation **Pad n** , where n is a digit, indicates the n key on the numeric keypad to the right of the main keyboard area.

Note: Unless otherwise noted, the functions initiated by a three-key **Ctrl-Alt-key** sequence require the **Alt** key on the *left* side of the standard RT keyboard. Functions initiated with **Alt-key** (or **Shift-key**) can be selected with either the left or the right **Alt** key (or **Shift** key).

AIX System Functions:

- | | |
|------------------|--|
| Alt-Pause | Sends the interrupt signal, SIGINT , to all AIX processes associated with the terminal (or virtual terminal) from which this key sequence is entered. This causes most processes to terminate, although a process can arrange to ignore or take other action on this signal. |
| Ctrl-V | Sends the quit signal, SIGQUIT , to all AIX processes associated with the terminal (or virtual terminal) from which this key sequence is entered. This causes most processes to terminate and produce a process image file in the current directory named core . However, a process can arrange to ignore or take other action on this signal. |

See "termio" on page 5-133 for additional AIX keystroke control sequences.

Virtual Terminal Functions:

- | | |
|---------------------|---|
| Alt-Action | Changes the active display screen to the next virtual terminal (if any). |
| Shift-Action | Changes the active display screen to the previous virtual terminal (if any). |
| Ctrl-Action | Changes the active display screen to the command virtual terminal (if defined). |

Coprocessor Functions:

These functions can be initiated with either the **Left** or the **Right Alt** key.

- | | |
|------------------------|--|
| Ctrl-Alt-Del | Re-IPLs the coprocessor (if configured). |
| Ctrl-Alt-Action | Exits the coprocessor direct mode. |

IPL (System Restart) Functions:

The following references to virtual machines apply to the kernel.

Ctrl-Alt-Home Terminates all virtual machines and re-IPLs the virtual machines that are configured to be IPLed automatically. If diskette drive #1 contains a valid virtual machine IPL diskette, then only that virtual machine is IPLed.

Ctrl-Alt-Pause Performs a soft re-IPL, reloading the VRM as well as the virtual machines that are configured to be IPLed automatically.

Warning: This IPL function does *not* terminate the virtual machines before reloading the system. Data can be lost if you do not shut down all virtual machines before pressing this IPL key sequence.

Ctrl-Alt-Pad6 Performs a power-on reset re-IPL. This runs the Power-On Self Test (POST), then reloads the VRM and the virtual machines that are configured to be IPLed automatically.

Warning: This IPL function does *not* terminate the virtual machines before reloading the system. Data may be lost if you do not shut down all virtual machines before pressing this IPL key sequence.

IPL Device Specification Functions:

The following key sequences set a value in NVRAM that identifies the *second* device to be read during a VRM IPL. The system first checks diskette drive #1 for a valid IPL record, then the second device, then each of the fixed disks.

Ctrl-Alt-A Designates diskette drive #1 as the second device to be read.

Ctrl-Alt-B Designates diskette drive #2 (if configured) as the second device to be read.

Ctrl-Alt-C Designates fixed-disk drive #1 as the second device to be read.

Ctrl-Alt-D Designates fixed-disk drive #2 (if configured) as the second device to be read.

Ctrl-Alt-E Designates fixed-disk drive #3 (if configured) as the second device to be read.

keyboard

System Dump Functions:

Note: Before attempting to use any of the following system dump key sequences, see *Software Problem Determination Guide* for more detailed information.

Ctrl-Alt-End	Performs a dump of the kernel.
Ctrl-Alt-Pad8	Performs a system dump of selected parts of real memory.
Ctrl-Alt-Pad7	Performs a dump of all real memory.

VRM Function:

Ctrl-Alt-Pad4	Invokes the VRM debugger.
----------------------	---------------------------

Related Information

In this book: “data stream” on page 4-5, “display symbols” on page 4-26, “hft” on page 5-39, “Set Keyboard Map (HFSKBD)” on page 5-53, and “termio” on page 5-133.

Keyboard Description and Character Reference.

Software Problem Determination Guide.

lp

Purpose

Supports the line printer device driver.

Synopsis

```
#include <sys/lprio.h>
```

Description

The **lp** driver provides an interface to the port used by a printer. If an adapter for a printer is not installed, an attempt to open fails. The **close** system call waits until all output completes before returning to the user. The **lp** driver allows only one process to write to a printer adapter at a time. If the printer adapter is busy, the **open** system call returns an error. However, the driver allows multiple **open** system calls to occur if they are **read-only**. Thus, the **splp** command can be run when the printer adapter is currently in use.

The **lp** driver interprets carriage returns, backspaces, line feeds, tabs, and form feeds depending on the modes that are set in the driver (via **splp**). The number of lines per page, columns per line, and the indentation at the beginning of each line can also be selected. The defaults are set at 66 lines per page, and 80 columns per line with no indenting.

ioctl Operations

Syntax for the enhanced control function is:

```
#include <sys/lprio.h>
ioctl (fildes,command,arg)
    int fildes;                /* file descriptor */
    int command;              /* command type */
    struct LPRUDE *arg;        /* pointer to info structure */
```

The possible command types and their descriptions are:

IOCINFO	Returns a structure defined in sys/devinfo.h , which describes the device.
IOCTYPE	Returns device LPR (line printer) defined in sys/devinfo.h .
LPRGET	Gets page length, width, and indent. This structure is defined in sys/lprio.h .

LPRGETA	Gets the RS232 parameters. These are the values for baud rate, character size, stop bits, and parity. Refer to the LPR232 structure and to the termio.h structure.
LPRGETV	Gets optional line printer modes. See the following LPRMODE structure.
LPRGMOD	<p>Gets the RT optional printer modes. These optional printer modes support the synchronous versus asynchronous write interface, as well as the report all errors versus wait until error correction error reporting mode. Refer to the following OPRMODE structure.</p> <p>The FONTINIT flag is initially off. It is turned on by an application when a printer font has been initialized. It is turned off when an application wants fonts to be reinitialized and by the lp device driver when a FATAL printer error occurs.</p>
LPRSET	Sets page length, width, and indent values. This structure is defined in sys/lprio.h .
LPRSETA	Sets the RS232 parameters. These are the values for baud rate, character size, stop bits, and parity. Refer to the LPR232 structure that follows and to the termio.h structure.
LPRSETV	Sets optional line printer modes. See the following LPRMODE structure.
LPRSMOD	<p>Sets the RT optional printer modes. These optional printer modes support the synchronous versus asynchronous write interface, as well as the report all errors versus wait until error correction reporting mode. Refer to the following OPRMODE structure.</p> <p>The FONTINIT flag is initially off. It is turned on by an application when a printer font has been initialized. It is turned off when an application wants fonts to be reinitialized and by the lp device driver when a FATAL printer error occurs.</p>
LPRUFLS	Flushes any data currently in progress and causes the printer to be initialized. Due to VRM queuing mechanism, printing of the data in the current queue element may take some time to complete before printing stops. This can be used during the course of normal print operations, or following an error indication.
LPRUGES	Gets the device driver error structure (LPRUDE). The <i>arg</i> parameter must be specified to point to the structure.
LPRURES	Resumes printing from the point of interruption following an error. If an error did not occur, then this control has no effect.
LPRVRMS	Sets VRM device-dependent parameters. This causes a set device characteristics Start-I/O SVC to be issued with the device-dependent parameters.

- LPRVRMG** Gets Virtual Resource Manager (VRM) device-dependent parameters. This returns the structure obtained by issuing a **Query_Device SVC**. The structure contains status information, hardware characteristics, device-dependent parameters, and RAS log information.
- LPRGTOV** Gets the current time out value and stores it in the **lptimer** structure pointed to by the *arg* parameter. The time out value is measured in seconds.
- LPRSTOV** Sets the time out value. The *arg* parameter points to a **lptimer** structure. The time out value must be given in seconds.

Most of these **ioctl** operations require the *arg* parameter to point to one of the following structures:

```
struct devinfo
{ char devtype; /* devtype for printer is 'l' */
  char flags;
};

/* used with LPRGET, LPRSET */
struct LPRIO {
    int ind;      /* indent value */
    int col;      /* maximum character count */
    int line;     /* maximum line count */
};

/* used with LPRGET, LPRSET */
struct LPRMODE {
    int modes; /* optional line printer modes */
};

/* bit definitions for the modes field in LPRMODE */
#define PLOT 01      /* if on, no interpretation of any character */
#define NOFF 0400    /* if on, simulate the form-feed function */
#define NONL 01000   /* if on, substitute carriage returns for */
                    /* any line-feeds */
#define NOTAB 02000  /* if on, don't expand tabs, else simulate */
                    /* 8 position tabs with spaces */
#define NOBS 04000   /* if on, no backspaces to the printer */
#define NOCR 010000  /* if on, substitute line-feeds for any */
                    /* carriage returns */
#define CAPS 020000  /* if on, map lower-case alphabets */
```

```
/* to upper case */
#define WRAP 040000 /* if on, print characters beyond the page */
/* width on the next line, instead of */
/* truncating */

struct OPRMODE {
    int flags; /* optional line printer modes */
};
#define SYNC 01 /* asynchronous is default. */
/* synchronous if on */
#define ALLERR 02 /* wait until error correction is default.*/
/* report all errors if on */
#define FONTINIT 04 /* file initialization */
struct LPRUDE /* device error-reporting structure */
{
    int status; /* error reason code */
    int cresult; /* current operation result :PSB */
    int tadapt; /* adapter type */
    int npio; /* number of pending IO operations */
};

/* status values - error reason codes */

struct LPR232 /* settings for RS232 */
{
    unsigned c_cflag; /* error reason code */
};

/* used with LPRGTOV and LPRSTOV */
struct lptimer
{
    unsigned v_timeout; /* timeout value in seconds */
}
```

File

/dev/lp*

Related Information

In this book: “ioctl” on page 2-407 and “devinfo” on page 3-66.

The **splp** command in *AIX Operating System Commands Reference*.

mem, . . .

mem, kmem, nvram

Purpose

Provides memory, kernel memory, and nonvolatile memory images.

Description

The **mem**, **kmem**, and **nvram** files are pseudo-device driver files. These device drivers are an image of physical memory in the system. These files can be used, for example, to examine or to patch the system.

The **mem** file is a special file that is an image of the system-generated virtual memory. It can be used, for example, to examine, and even to patch the system. You should use the **readx** and **writex** system calls to read or write **mem**. The parameter is the segment ID of the memory segment to be read or written. The system uses the seek offset as the byte offset within the segment. The high-order 4 bits of the seek offset are ignored. If there is a segment ID of 0, or if you use **read** instead of **readx**, **mem** acts like **kmem**.

The file **kmem** is similar to **mem**, except it is used to access kernel and calling program virtual memory. The seek offset is an address in kernel virtual memory. Seek offsets of less than 0x1000 0000 address the kernel segment, which contains both text and data. Seek offsets from 0x1000 0000 to 0x1FFF FFFF address the text segment of current process and so on.

The **nvram** file is used to access the system nonvolatile memory. It contains an area 16 bytes long to log RT errors when the system cannot make a permanent copy of errors on the disk. Unlike **mem** and **kmem**, information written to this device is retained after power is removed from the system.

An invalid virtual address or segment ID used with **mem**, **kmem**, or **nvram** causes errors to be returned.

Files

/dev/mem
/dev/kmem
/dev/nvram

null

Purpose

Provides a null device.

Description

The **null** file is a pseudo-device driver file with no associated hardware. Data written to this file is discarded. Reads from this file always return 0 bytes. Use this file to read or write null data as required.

File

/dev/null

Purpose

Provides the interface to AIX messages.

Description

The **osm** driver collects system messages provided by the AIX kernel and application programs. These system messages are available to a daemon reading this file. System messages have two sources:

- The AIX kernel provides messages by calls to the kernel **printf** routine.
- Application programs open and write to this file.

Operating system messages are stored in a circular buffer in the system and can be read or written using the **osm*** special files. A read from **osm*** files returns some portion of the data in the circular buffer. A write to the files adds user data to the current end of the circular buffer. Any number of users may use **osm*** files in the same instance of time.

Read operations from the **osm** file start at the current end of the circular buffer and wait for new data to be added. Read operations from the file **/dev/osm.curr** start at the beginning of the circular buffer and return 0 bytes when the current end of the buffer is reached. Read operations from the **/dev/osm.all** file start at the beginning of the circular buffer, go to the current end of the circular buffer, and wait for new data to be added.

File

/dev/osm*

Related Information

In this book: “**rasconf**” on page 3-189.

prf

Purpose

Profiles the kernel.

Description

The **prf** file is a pseudo-device driver file with no associated hardware. This device driver file provides access to activity information in the kernel.

This file is initialized to contain an array of sorted kernel text addresses. An **ioctl** system call issued with command value 3 and parameter value equal to 1 starts monitoring operations. Each subsequent clock interrupt causes the table of addresses to be searched and the highest address less than or equal to the program counter at the time of the interrupt to be located. The counter corresponding to the located address is incremented. An interrupt that occurs while in user mode, increments a miscellaneous counter. An **ioctl** system call issued with command value 3 and parameter value equal to 0 stops the monitoring.

Reading this file returns the array of addresses and the array of counters. The miscellaneous counter is returned last. The information is returned in a single **read** system call. The buffer supplied must be large enough to hold the information.

You can determine the status of the profiling facility, by issuing an **ioctl** system call with command value 1. Bit 1 (the least significant bit) of the return value is set if monitoring is on. Bit 2 is set if a valid list of text addresses was loaded. An **ioctl** system call issued with a command value 2 returns the number of loaded text addresses.

File

/dev/prf

Related Information

The **config** and **profiler** commands in *AIX Operating System Commands Reference*.

Purpose

Implements a pseudo-terminal device.

Synopsis

```
#include <sys/devinfo.h >
#include <sys/pty.h >
#include <sys/tty.h >
```

Description

A **pty** device is a pair of bidirectional character device drivers that implement a pseudo-terminal. A pseudo-terminal can act as a keyboard and a display to existing software that uses the standard terminal device interface described in “termio” on page 5-133. This is useful for a variety of applications such as a remote login facility or a windowing system.

Each pseudo-terminal (or **pty**) consists of two device drivers called a controller and a server. The **server** or **server side** of a **pty** has a standard terminal interface that can support a login shell or other software that normally communicates with terminals. The **controller** or **controller side** of a **pty** interfaces with software that generates and receives data as if it were a user at a terminal. Data written to the controller is passed directly to the server, which is then read and processed as if entered from a keyboard. Data written to the server (as if to be displayed on a terminal screen) is passed directly to the controller.

The corresponding special files are named **/dev/ptcn** for the controller and **/dev/ptsn** for the server, where *n* is a decimal number. A one-to-one correspondence exists between each controller-server pair with names that end in the same number. For example, **/dev/ptc0** and **/dev/pts0** together form a **pty**.

To assist in porting Berkeley applications, the appropriate BSD names link to the appropriate AIX names. For example, **/dev/ptyp0** links to **/dev/ptc0** and **/dev/ttyp0** links **/dev/pts0**.

The **ptybuffers** keyword in the **/etc/master** file controls the number of **ptys** that can be present in the system. The maximum number of **ptys** is 256, while the default number set by the **/etc/master** file is 16.

For a remote login application such as Telnet, use the **devices** command to add the server side of a **pty** to the system configuration and enable it as a login port. In the case of

Telnet, the **telnetd** daemon opens the controller side of the **pty** and passes data sent over the network to the login shell.

When using a **pty** for applications other than remote login, a program must take into account the fact that a logger process may have already issued an **open** to the server side of the **pty**. When a logger opens the server side, the **open** system call suspends the process to wait for another process to open the controller side. Use the following strategy to detect this situation:

1. Open **/dev/ptcn**.
2. Issue an **ioctl** system call to perform the **PTYSTATUS** operation.
3. If the status indicates that the server side has already been opened, then close the **pty** controller and try a **/dev/ptcn** device with a different value for *n*.
4. If the status indicates that the server side has not been opened, then open the corresponding **/dev/ptsn** device.

Note: The server side of a **pty** can be opened multiple times, but the controller can be opened only once. Attempting to open the controller side more than once causes an error.

select Support

The **pty** device driver supports the **select** system call in the following manner:

- Read selects are satisfied when input data is available.
- Write selects are satisfied when data can be accepted.
- Exception selects are never satisfied, or hang indefinitely if no *timeout* value is specified.

See “select” on page 2-703 for more information about this system call.

ioctl Operations

The interface to the server side of the **pty** device is identical to the standard interface for terminals, which is described in “termio” on page 5-133.

The controller side of the **pty** device driver supports the following **ioctl** operations. (See “ioctl” on page 2-407 for detailed information about the **ioctl** system call.)

IOCTYPE	Returns the device type DD_PSEU to indicate that this is a pseudo-terminal device. This operation ignores the <i>arg</i> parameter.
IOCINFO	Copies the devinfo structure for the device into the buffer pointed to by the <i>arg</i> parameter passed to ioctl . See “devinfo” on page 3-66 for details about this structure.

PTYSTATUS	Returns the state of the pty , which is composed of two halfwords. The upper half contains the number of opens currently outstanding against the controller, and the lower half contains the number of opens currently outstanding against the server. This operation ignores the <i>arg</i> parameter.						
PTYIOR	Reports the number of characters available to be read. The <i>arg</i> parameter is a pointer to an integer, into which this value is stored.						
PTYIOW	Reports the number of characters on the raw and canonical queues. The <i>arg</i> parameter is a pointer to an integer, into which this value is stored.						
PTYGETM	Gets the current mode of the pty . The <i>arg</i> parameter is a pointer to an integer, into which the mode is stored. See the description PTYSETM for an explanation of the possible mode values.						
PTYSETM	Sets the current mode of the pty . The <i>arg</i> parameter is a pointer to an integer that contains the mode to be set. The mode is zero or more of the following values logically ORed together: <table><tr><td>RAWQINT</td><td>Sends the SIGPTY signal to the process when enough buffer space is available for writing to the pty.</td></tr><tr><td>OUTQINT</td><td>Sends the SIGPTY signal to the process when data is available to be read.</td></tr><tr><td>REMOTE</td><td>Controls the flow of input to the pty, but does not edit the input. In other words, START and STOP (Ctrl-S and Ctrl-Q) controls are processed, but no editing is done on the data stream (such as ERASE, KILL, or ICRNL).</td></tr></table>	RAWQINT	Sends the SIGPTY signal to the process when enough buffer space is available for writing to the pty .	OUTQINT	Sends the SIGPTY signal to the process when data is available to be read.	REMOTE	Controls the flow of input to the pty , but does not edit the input. In other words, START and STOP (Ctrl-S and Ctrl-Q) controls are processed, but no editing is done on the data stream (such as ERASE , KILL , or ICRNL).
RAWQINT	Sends the SIGPTY signal to the process when enough buffer space is available for writing to the pty .						
OUTQINT	Sends the SIGPTY signal to the process when data is available to be read.						
REMOTE	Controls the flow of input to the pty , but does not edit the input. In other words, START and STOP (Ctrl-S and Ctrl-Q) controls are processed, but no editing is done on the data stream (such as ERASE , KILL , or ICRNL).						
PTYADDM	Adds to the current pty mode by logically ORing the specified value with the existing mode. The <i>arg</i> parameter is a pointer to an integer that contains the mode bits to be set. See the description PTYSETM for an explanation of the possible mode values.						
PTYDELM	Deletes from the current pty mode. The <i>arg</i> parameter is a pointer to an integer. The bits that are set in this integer specify the mode bits to be turned off. See the description PTYSETM for an explanation of the possible mode values.						
TIOCPKT	Enables or disables packet mode. The <i>arg</i> parameter is a pointer to an integer that contains the packet mode to be enabled or disabled. The mode is zero or more of the following values logically ORed together: <table><tr><td>FLUSHREAD</td><td>Indicates when the terminal read queue is flushed.</td></tr><tr><td>FLUSHWRITE</td><td>Indicates when the terminal write queue is flushed.</td></tr><tr><td>STOP</td><td>Indicates output to the terminal is stopped with STOP (Ctrl-S) control.</td></tr></table>	FLUSHREAD	Indicates when the terminal read queue is flushed.	FLUSHWRITE	Indicates when the terminal write queue is flushed.	STOP	Indicates output to the terminal is stopped with STOP (Ctrl-S) control.
FLUSHREAD	Indicates when the terminal read queue is flushed.						
FLUSHWRITE	Indicates when the terminal write queue is flushed.						
STOP	Indicates output to the terminal is stopped with STOP (Ctrl-S) control.						

START	Indicates output to the terminal has restarted.
DOSTOP	Indicates packet mode is stopped when START and STOP (Ctrl-S and Ctrl-Q) controls are processed.
NOSTOP	Indicates packet mode is stopped if the START and STOP controls are not Ctrl-S and Ctrl-Q .

Diagnostics

System calls to a **pty** device fail and set **errno** to indicate the error if one or more of the following are true:

EINVAL	An invalid parameter was encountered, such as a negative number of bytes to be written.
ENXIO	The pty cannot be opened because the pty number is out of range.
EIO	A read , write , or ioctl operation was attempted that requires both sides of the pty to be open, making a complete connection.
EACCES	An attempt was made to open the controller side of a pty more than once.

Files

/dev/ptc0, /dev/ptc1, . . . Controller Devices.
/dev/pts0, /dev/pts1, . . . Server Devices.

Related Information

In this book: “**ioctl**” on page 2-407, “**open**” on page 2-538, “**master**” on page 3-125, “**ports**” on page 3-173, “**system**” on page 3-210, and “**fcntl.h**” on page 4-58.

The **tn** and **telnetd** commands in *Interface Program for use with TCP/IP*.

The **devices** and **init** commands in *AIX Operating System Commands Reference*.

tape

tape

Purpose

Supports the sequential access bulk storage medium device driver.

Description

Magnetic tapes are used primarily for backups, file archives, and other off-line storage. Tapes are accessed through the special files **rmt0**, . . . , **rmt15**. The **r** indicates “raw,” which indicates access through the character special interface. A streaming tape does not lend itself well to the category of a block device; only these character interface files are provided. The number following the **rmt** is the minor device number. The two low-order bits of the minor device number select the transport. If the third bit (04 octal or 0x04) is set, the driver does not rewind the tape after it is closed. If the fourth bit (010 octal or 0x08) is set, the tape is retensioned (wound completely forward and then rewound) after it is opened and before any other operations.

On a system with a single tape drive, **/dev/rmt0** does not retension the tape, but does rewind it on close. **/dev/rmt4** (bits = 0100) does not perform any special actions on open or close. **/dev/rmt8** (bits = 1000) retensions the tape and rewinds it on close; and **/dev/rmt12** (bits = 1100) retensions the tape on open, but does not rewind.

When opened for reading or writing, the tape is assumed to be positioned as desired. When the tape opens and writes to a file, a single tape mark is written if the file is no rewind on close, while a double tape mark is written if the tape is to be rewound. If the file is no rewind and opened read only, the tape is positioned after the end of the file (**EOF**) following the data just read. Once opened, reading is restricted to between the position when opened and the next **EOF**. By specifically choosing **rmt** files, it is possible to read and write multiple-file tapes.

Each **read** or **write** call reads or writes the next record on the tape. The record written by **write** is the same length as the buffer given. During a **read**, the record size is returned as the number of bytes read, up to the buffer size specified. Seeks are ignored. An **EOF** is returned as a zero-length read, with the tape positioned before the **EOF**.

A number of **ioctl** operations are available. In addition to **IOCTYPE** and **IOCINFO** types, the following **ioctl** calls are defined.

The parameter to the **ioctl** system call using the **STIOCTOP** command is the address of a **stop** structure, which contains the following members:

```
short    st_op;        /* Streaming tape operation */
daddr_t  st_count;     /* Number of times to perform */
```

The **st_op** operation is performed **st_count** times, except where it is not logical to do so (for example, rewind). The operations available are:

```
#define STRESET 5      /* reset device */
#define STREW 6        /* rewind */
#define STERASE 7      /* erase tape, retention, leave at load point */
#define STREten 8      /* erase tape, retention, leave at load point */
#define STWEOF 10      /* write an end-of-file record */
#define STFSF 11       /* forward space file */
#define STFSR 13       /* forward space record */
#define STRAS1 15      /* drive self test 1 */
#define STRAS2 16      /* drive self test 2 */
#define STRAS3 17      /* drive self test 3 */
                        /* this test needs an */
                        /* erased write-protected tape */
```

The status of a tape drive can be determined by issuing the following **STIOCGET** **ioctl** system call:

```
/* structure for STIOCGET - streaming tape get status command */
struct stget {
    short st_type;      /* type of device */
    struct dsreg {
        unsigned short ds_dstat; /* drive status */
        unsigned short ds_soft;  /* soft error count */
        unsigned short ds_under; /* underrun count */
        unsigned char  ds_rcom;  /* command received by adapter */
        unsigned char  ds_blk;   /* adapter block count */
        unsigned char  ds_rstat; /* status register */
    };
};
```

tape

```
        unsigned char  ds_code;    /* adapter completion code */
        unsigned char  ds_lcom;    /* last command given to adapter */
        unsigned char  ds_lstcom;  /* last streaming tape device */
                                   /* drive command */
        unsigned char  ds_res[4]   /* reserved */
    } st_dsreg;
};
/*
 * Constants for st_type byte - ST_SST streaming tape
 */
```

In addition to those errors listed in **ioctl**, **open**, **read**, and **write**, system calls against this device fail in the following circumstances:

- EINVAL** **O_APPEND** is supplied as a mode in which to open.
- EINVAL** A write attempt while the tape is in read mode, or a read attempt while the tape is in write mode.
- EINVAL** A **count** parameter to **read** or **write** is not 0, modulo 512.
- EIO** A parameter to **ioctl** is not allowed in the current streaming mode.
- ENXIO** The tape is write-protected or there is no tape in the drive.

Note: The streaming tape device driver has a concept of current “streaming mode.” Therefore, many operations are invalid most of the time. In particular, no reads are allowed after an initial write or writes allowed after an initial read. You must wait until the device is reset either by closing a rewind-on-close special file, or by the **tctl** command.

File

/dev/rmt*

Related Information

In this book: “**ioctl**” on page 2-407, “**open**” on page 2-538, “**read**, **readx**” on page 2-591, and “**write**, **writex**” on page 2-877.

The **tctl** command in *AIX Operating System Commands Reference*.

termio

Purpose

Provides the general terminal interface.

Synopsis

```
#include <sys/hft.h>
#include <sys/termio.h>
#include <sys/tty.h>
```

Description

All of the asynchronous communications ports use the same general interface, regardless of the hardware used.

When a terminal file is opened, it normally causes the process to wait until a connection is established. In practice, user programs seldom open these files. They are opened by **getty** and become standard input, output, and error files for a user. The first terminal file not already associated with a process group that is opened by the process group leader becomes the **control terminal** for that process group. The control terminal plays a special role in handling **QUIT** and **INTERRUPT** signals as discussed later. During a **fork** system call, the child process inherits the control terminal. A process can break the association to the group using the **setpgrp** system call.

A terminal associated with one of these files ordinarily operates in full-duplex mode. Characters can be typed at any time, even while output is occurring. These characters can be lost, however, when the input buffers become completely full or when the user accumulates the maximum number of input characters allowed that were not read by a program. Currently, this limit is 256 characters. When the input limit is reached, all the saved characters are erased from the input buffer without notice.

Normally, terminal input is processed in units of lines. A line is delimited by a new-line (ASCII LF) character, an end-of-file (ASCII EOT) character, or an end-of-line character. This means that a program attempting to read is suspended until an entire line is typed. Also, no matter how many characters are requested in the read call, at most one line is returned. It is not, however, necessary to read a whole line at once. Any number of characters can be requested in a read without losing information.

During input, **erase** and **kill** processing is performed normally. By default, the **Ctrl-H** character erases the last character typed, but does not erase beyond the beginning of the line. By default, the **Ctrl-U** character kills (deletes) the entire input line, and optionally outputs a new-line character. Both these characters operate on a keystroke basis

independently of any backspacing or tabbing that was done. Both the erase and kill characters can be entered literally by preceding them with the \ (backslash) escape character. In this case, the escape character is not read. The erase and kill characters can be changed.

Certain characters have special functions on input. These functions and their default character values are summarized as follows:

- EOF** **Ctrl-D** or ASCII EOT is used to generate an end-of-file character from a terminal. When received, all the characters waiting to be read are immediately passed to the program, without waiting for a new-line character, and the EOF is discarded. Thus, if there are not any characters waiting (indicating the EOF occurred at the beginning of a line), zero characters are passed back, which is the standard end-of-file indication.
- EOL** ASCII NUL is an additional line delimiter, like NL. It is not normally used.
- ERASE** **Ctrl-H** erases the preceding character. It does not erase beyond the start of a line, as delimited by an NL, EOF, or EOL character.
- INTR** **Rubout** or ASCII DEL (**Ctrl-Backspace** on the RT keyboard) generates a **SIGINT** (interrupt) signal, which is sent to all processes with the associated control terminal. Normally, each such process is forced to terminate, but arrangements can be made either to ignore the signal or to receive a trap to an agreed-upon location. See "signal" on page 2-750.
- KILL** **Ctrl-U** deletes the entire line, as delimited by an NL, EOF, or EOL character.
- NL** ASCII LF is the normal line delimiter. It cannot be changed or escaped.
- QUIT** **Ctrl-V** or ASCII SYN generates a **QUIT** signal. Its treatment is identical to the interrupt signal except that, unless a receiving process made other arrangements, it is not only terminated but a memory file (called **core**) is created in the current working directory.
- START** **Ctrl-Q** or ASCII DC1 is used to resume output that was suspended by a STOP character. While output is not suspended, START characters are ignored and not read. The start/stop characters cannot be changed or escaped.
- STOP** **Ctrl-S** or ASCII DC3 is used to temporarily suspend output. It is useful with terminals that have displays to prevent output from disappearing before it can be read. While output is suspended, STOP characters are ignored and not read.

The character values for INTR, QUIT, ERASE, KILL, EOF, and EOL can be changed to suit individual preferences. The ERASE, KILL, and EOF characters can be escaped by a preceding \ (backslash) character, in which case the special function is not done.

When the carrier signal from the dataset drops, a **hangup** signal (**SIGHUP**) is sent to all processes that have this terminal as the control terminal. Unless other arrangements were made, this signal causes the process to terminate. If the hangup signal is ignored, any

subsequent read returns with an end-of-file indication. Thus, programs that read a terminal and test for the end of the file can terminate appropriately.

When one or more characters are written, they are transmitted to the terminal as soon as previously written characters finish typing. Input characters are usually echoed by putting them in the output queue as they arrive, but see “Enhanced Edit Mode” on page 5-141. If a process produces characters more rapidly than they can be typed, it is suspended when its output queue exceeds some limit. When the output decreases to a determined threshold, the program is resumed.

Several **ioctl** system calls apply to terminal files. The primary calls use the following structures defined in the **termio.h** header file:

```
#define NCC 8
struct termio {
    unsigned short  c_iflag; /* input modes */
    unsigned short  c_oflag; /* output modes */
    unsigned short  c_cflag; /* control modes */
    unsigned short  c_lflag; /* local modes */
    char            c_line;  /* line discipline */
    unsigned char    c_cc[NCC]; /* control chars */
};

struct tty_page {
    char tp_flags;
    unsigned char tp_slens;
};
```

The special control characters are defined by the **c_cc** array. The relative positions and initial values for each function are as follows:

c_cc[0]	INTR	Ctrl-Backspace (DEL)
c_cc[1]	QUIT	Ctrl-V (SYN)
c_cc[2]	ERASE	Backspace (BS)
c_cc[3]	KILL	Ctrl-U (NAK)
c_cc[4]	EOF	Ctrl-D (EOT)
c_cc[5]	EOL	Ctrl-@ (NUL)
c_cc[6]	reserved	
c_cc[7]	reserved	

termio

The **c_iflag** field describes the basic terminal input control. The initial input control value is all bits clear. The possible values are:

IGNBRK	0000001	Ignore break condition.
BRKINT	0000002	Signal interrupt on break.
IGNPAR	0000004	Ignore characters with parity errors.
PARMRK	0000010	Mark parity errors.
INPCK	0000020	Enable input parity check.
ISTRIP	0000040	Strip character.
INLCR	0000100	Map new-line character (NL) to carriage return character (CR) on input.
IGNCR	0000200	Ignore carriage return character.
ICRNL	0000400	Map carriage return character to new-line character on input.
IUCLC	0001000	Maps uppercase to lowercase on input.
IXON	0002000	Enables start/stop output control.
IXANY	0004000	Enables any character to restart output.
IXOFF	0010000	Enables start/stop input control.
ASCEDIT	0020000	Enables enhanced editing on ASCII terminals.

The values in this field are described as follows:

IGNBRK	If set, the break condition (a character framing error with data all zeros) is ignored. It is not put on the input queue and therefore not read by any process. Otherwise, if BRKINT is set, the break condition generates an interrupt signal and flushes both the input and output queues. If IGNPAR is set, characters with other framing and parity errors are ignored.
PARMRK	If set, a character with a framing or parity error that is not ignored is read as the 3-character sequence: 0377, 0, <i>x</i> , where <i>x</i> is the data of the character received in error. If ISTRIP is not set, then a valid character of 0377 is read as 0377, 0377 to avoid ambiguity. If PARMRK is not set, a framing or parity error that is not ignored is read as the character NUL (0).
INPCK	If set, input parity checking is enabled. If not set, input parity checking is disabled. This allows output parity generation without input parity errors.
ISTRIP	If set, valid input characters are first stripped to 7 bits; otherwise all 8 bits are processed.
INLCR	If set, a received new-line character is translated into a carriage-return character. If IGNCR is set, a received carriage-return character is ignored (not read). If ICRNL is set, a received carriage-return character is translated into a new-line character.
IUCLC	If set, a received uppercase alphabetic character is translated into the corresponding lowercase character.

IXON	If set, start/stop output control is enabled. A received STOP character suspends output and a received START character restarts output. All start/stop characters are ignored and not read. If IXANY is set, any input character restarts output that was suspended.
IXOFF	If set, the system transmits START/STOP characters when the input queue is nearly empty or full.
ASCEDIT	If set, ASCII keyboards can be used to enter enhanced edit line discipline commands.

The **c_oflag** field specifies how the system treats output. The initial output control value is all bits clear.

OPOST	0000001	Postprocess output.
OLCUC	0000002	Map lowercase to uppercase on output.
ONLCR	0000004	Map new-line character to CR-NL on output.
OCRNL	0000010	Map carriage-return to new-line on output.
ONOCR	0000020	No carriage-return character output at column 0.
ONLRET	0000040	Perform carriage return function using new-line character.
OFILL	0000100	Use fill characters for delay.
OFDEL	0000200	Fill is DEL or NUL.
NLDLY	0000400	Select new-line character delays:
NL0	0	
NL1	0000400	
CRDLY	0003000	Select carriage-return delays:
CR0	0	
CR1	0001000	
CR2	0002000	
CR3	0003000	
TABDLY	0014000	Select horizontal-tab delays:
TAB0	0	
TAB1	0004000	
TAB2	0010000	
TAB3	0014000	Expand tabs to spaces.
BSDLY	0020000	Select backspace delays:
BS0	0	
BS1	0020000	
VTDLY	0040000	Select vertical-tab delays:
VT0	0	
VT1	0040000	
FFDLY	0100000	Select form-feed delays:
FF0	0	

FF1 0100000

OPOST If set, output characters are post-processed as indicated by the remaining flags; otherwise characters are transmitted without change.

OLCUC If set, a lowercase alphabetic character is transmitted as the corresponding uppercase character. This function is often used in conjunction with **IUCLC**.

ONLCR If set, the new-line character is transmitted as the carriage-return new-line character pair.

OCRNL If set, the carriage-return character is transmitted as the new-line character.

ONOCR If set, no carriage-return character is transmitted when at column 0 (first position).

ONLRET If set, the new-line character is assumed to do the carriage return function. The column pointer is set to 0 and the delay specified for carriage return is used. Otherwise the new-line character is assumed to do just the line feed function; the column pointer remains unchanged. The column pointer is also set to 0 if the carriage-return character is actually transmitted.

OFILL If set, fill characters are transmitted for delay instead of a timed delay. This is useful for high baud rate terminals that need only a minimal delay.

OFDEL If set, the fill character is DEL, otherwise NUL.

NLDLY, CRDLY, TABDLY, BSDLY, VTDLY, FFDLY

The delay bits specify how long transmission stops to allow for mechanical or other movement when certain characters are sent to the terminal. In all cases, a value of 0 indicates no delay. If **ONLRET** is set, the carriage return delays are used instead of the new-line delays.

TAB3 If set, specifies that tabs are to be expanded into spaces.

The **c_cflag** field describes the hardware control of the terminal:

CBAUD	0000017	Baud rate
B0	0	Hang up
B50	0000001	50 baud
B75	0000002	75 baud
B110	0000003	110 baud
B134	0000004	134.5 baud
B150	0000005	150 baud
B200	0000006	200 baud

B300	0000007	300 baud
B600	0000010	600 baud
B1200	0000011	1200 baud
B1800	0000012	1800 baud
B2400	0000013	2400 baud
B4800	0000014	4800 baud
B9600	0000015	9600 baud
EXTA	0000016	External A
EXTB	0000017	External B
CSIZE	0000060	Character size:
CS5	0	5 bits
CS6	0000020	6 bits
CS7	0000040	7 bits
CS8	0000060	8 bits
CSTOPB	0000100	Send 2 stop bits, else one.
CREAD	0000200	Enable receiver.
PARENB	0000400	Parity enable.
PARODD	0001000	Odd parity, else even.
HUPCL	0002000	Hang up on last close.
CLOCAL	0004000	Local line, else dial-up.

CBAUD These bits specify the baud rate. The zero baud rate, B0, is used to hang up the connection. If B0 is specified, the data-terminal-ready signal is not dropped. Normally, this disconnects the line. For any particular hardware, impossible speed changes are ignored.

CSIZE These bits specify the character size in bits for both transmit and receive. This size does not include the parity bit, if any. If CSTOPB is set, 2 stop bits are used; otherwise 1 stop bit is used. For example, at 110 baud, 2 stop bits are required.

CREAD If set, the receiver is enabled. Otherwise characters are not received.

PARENB If set, parity generation and detection is enabled and a parity bit is added to each character. If parity is enabled, the **PARODD** flag specifies odd parity if set; otherwise even parity is used.

The initial hardware control value after open is B300, CS8, CREAD, HUPCL.

HUPCL If set, the line is disconnected when the last process that has the line open, either closes it or the process terminates. That is, the data-terminal-ready signal drops.

CLOCAL If set, the line is assumed to be local, direct connection with no modem control. Otherwise modem control is assumed.

termio

The `c_lflag` field of the parameter structure is used by the line discipline to control terminal functions. The basic line discipline (0) provides the following:

ISIG	0000001	Enable signals.
ICANON	0000002	Canonical input (erase and kill processing).
XCASE	0000004	Canonical upper/lower presentation.
ECHO	0000010	Enable echo.
ECHOE	0000020	Echo erase character as BS-SP-BS.
ECHOK	0000040	Echo new-line character after kill character.
ECHONL	0000100	Echo new-line character.
NOFLSH	0000200	Disable flushing the queue after interrupt or quit.

ISIG If set, each input character is checked against the special control characters `INTR` and `QUIT`. If a character matches one of these control characters, the function associated with that character is performed. If `ISIG` is not set, checking is not done. Thus, these special input functions are possible only if `ISIG` is set. These functions may be disabled individually by changing the value of the control character to an unlikely or impossible value (for example, 0377 octal or 0xFF).

ICANON If set, canonical processing is enabled. Canonical processing enables the erase and kill edit functions, and the assembly of input characters into lines delimited by `NL`, `EOF`, and `EOL`. If `ICANON` is not set, then read requests are satisfied directly from the input queue. In this case, a read request is not satisfied until either at least `MIN` characters have been received, or the time out value `TIME` has expired since the last character was received. This allows bursts of input to be read, while still allowing single-character input. The `MIN` and `TIME` values are stored in the positions for the `EOF` and `EOL` characters, respectively. The time value represents tenths of seconds.

XCASE If set along with `ICANON`, an uppercase letter (or the uppercase letter translated to lowercase by `IUCLC`) is accepted on input by preceding it with a `\` (backslash) character, and is output preceded by a `\` (backslash) character. In this mode, the output generates and the input accepts the following escape sequences:

For: Use:

<code>\</code>	<code>\'</code>
<code> </code>	<code>\!</code>
<code>~</code>	<code>\^</code>
<code>{</code>	<code>\(</code>
<code>}</code>	<code>\)</code>
<code>\</code>	<code>\\</code>

For example, `A` is input as `\a`, `\n` as `\\n`, and `\N` as `\\\n`.

- ECHO** If set, characters are echoed as received. When ICANON is set, the following echo functions are possible. If ECHO and ECHOE are set, the erase character is echoed as ASCII BS SP BS, which clears the last character from a cathode-ray-tube screen. If ECHOE is set and ECHO is not set, the erase character is echoed as ASCII SP BS. If ECHOK is set, the new-line character is echoed after the kill character to emphasize that the line is deleted. Note that an escape character preceding the erase or kill character removes any special function. If ECHONL is set, the new-line character will be echoed even if ECHO is not set. This is useful for terminals set to local echo (sometimes called half duplex). Unless escaped, the EOF character is not echoed. Because EOT is the default EOF character, this prevents terminals that respond to EOT from hanging up.
- NOFLSH** If set, the normal flushing of the input and output queues associated with the quit and interrupt characters is not done.

Enhanced Edit Mode

The **c_line** field describes the line-discipline control value. The initial line-discipline control value is all bits clear. When **c_line** is equal to 1, it sets enhanced edit mode. This terminal line discipline provides a simple, line-oriented editing facility modeled on DOS Services.

The enhanced edit line discipline supports the same flags in **c_lflag** as the basic line discipline, but has the differences described following. The line discipline itself can be used from any terminal. To set the terminal to this mode from the shell, use the **stty** command. From a program, use the **ioctl** system call.

In enhanced edit mode, a special character buffer called the *template* is associated with the terminal. Using special function keys, the next line entered can be constructed out of the template and new characters.

Initially the template is blank. When a line is read by a running program, that line becomes the active template. The template can also be explicitly read and set by the application program using the **ioctl** system call and by the user with the **F5** key. The template does not directly appear on the terminal, but can be inspected by use of the function keys. When **Enter** is pressed, the old template is stacked and the current line becomes the active template. Up to eight templates can be stacked. The oldest template is deleted when a template is added to a full stack. The **↑ (Cursor Up)** and **↓ (Cursor Down)** keys change the active template from current template to the next or previous template respectively. The user can scroll through the stack, which automatically wraps back to the beginning when the end is reached.

Characters typed are not echoed to the terminal until the application program has issued a **read** system call to process it. The ERASE character is echoed as follows: when an ASCII TAB is deleted, the cursor is moved to the position where the TAB was typed, and the cursor will not be moved to the left of where input began on the current line.

Whenever nonprinting characters are directly echoed to the terminal, they appear as ^X, where X is the printable character that is 64 greater in value than the nonprinting character originally entered. Thus **Ctrl-A** is echoed as ^A, and so on. Finally, because DOS Services itself lacks any convention for escaping special characters, there is no way to specify literal occurrences of ERASE, KILL, or EOF. The \ (backslash) has no special meaning when used preceding them, and no escape character is defined.

Both the line buffer and the template can hold at least 128 characters. If the line buffer fills up, an audible signal sounds upon receipt of further characters, which are otherwise ignored. Exceptions are the **Backspace** and ← keys, which delete characters from the line buffer; and the **Esc**, **F5**, and **Enter** keys, which reset the line buffer after performing their functions.

While this mode is intended primarily for use with the **DOS Services** commands, it can be set at any time, although it may be overridden by programs such as editors that change the terminal characteristics in other ways.

To understand the use of the special function keys, it is helpful to consider their effect on the current line buffer, the template index, which points to a character in the template.

To use enhanced edit mode from an ordinary ASCII terminal, a compatibility mode must be set.

Native Keyboard	ASCII Keyboard	Action
Displayable Character	Displayable Character	Places the character typed in the line buffer. Advances the template buffer unless insert mode is on. (When insert mode is on, characters typed are effectively “inserted” into the line within the template. When off, characters typed effectively overwrite characters in the template.)
Return or Enter	Return or Enter	Sends the line buffer (as it appears on the screen) to the application program and accepts it as the active template. The line on the screen is advanced and the old template is placed on the stack. The line buffer is emptied and the template index reset to the beginning of the template.
Esc	Esc-Esc	Cancels the line currently being edited. A \ (backslash) character is echoed, then the cursor is moved to the same column on the next line as it was on the current line. The line buffer is emptied, and the template index is reset to the beginning of the template.
Del	Esc-D	Advances the template index by one. There is no effect on the screen. This, in effect, deletes the next template character, although Backspace restores it.

Native Keyboard	ASCII Keyboard	Action
↑	Esc-H	Changes the active template to the next one on the stack and copies the new active template to the line buffer. Advances the template index to the end of the template.
Ins	Esc-I	Sets insert mode on. Insert mode is reset by ↑, ↓, F1 , F2 , F3 , Esc , and Enter . When insert mode is set, inserts typed characters into the line within the template. When not set, characters that are typed overwrite characters in the template.
Backspace or ←	Backspace or Esc-J	Backspaces and removes a character from the screen, moving the template index back one, but not beyond the column where text entry began. If insert mode is not set, move the template index back but beyond the beginning of the template. Backspacing across a tab character moves the cursor to the position of the character before the tab character was typed.
↓	Esc-L	Changes the active template to the previous one in the stack and copies the new active template to the line buffer. Advances the template index to the end of the template.
F1 or →	Esc-K or Esc-1	Copies the character indicated by the template index to the line buffer and displays it. Advances the template index.
F2	Esc-2	Copies characters (as F1 does) up to, but not including, the next character typed.
F3	Esc-3	Copies to the line buffer, the characters in the template from the template index to the end of the template. Advances the template index to the end of the template.
F4	Esc-4	Advances the template index without copying characters (like Del) up to the next character typed. F4 is to Del what F2 is to F1 .
F5	Esc-5	Accepts the current line buffer as a new template. The @ character is echoed, and then the cursor is moved to the next line at the same column as that at which it started on the current line. The template index is reset to the beginning of the template.

Native Keyboard	ASCII Keyboard	Action
F6	Esc-6	Enters a Ctrl-Z character into the line buffer, as if it had been typed directly from the keyboard.
F7	Esc-7	Enters an ASCII NUL character into the line buffer, as if it had been typed directly from the keyboard.

Any other character typed is placed in the line buffer, except when ICANON is set. When ICANON is set, the special characters it provides are not retained in the buffer. The template index is advanced whenever a character is placed in the line buffer, unless insert mode is enabled.

The system console has other specific modes that are not valid for general terminal interfaces. See “hft” on page 5-39 for details.

select Support

The asynchronous terminal device driver supports the **select** system call in the following manner:

- Read selects are satisfied when input data is available.
- Write selects are always satisfied immediately.
- Exception selects are never satisfied, or hang indefinitely if no *timeout* value is specified.

See “select” on page 2-703 for more information about this system call.

ioctl Operations

The primary **ioctl** system calls have the format:

```
ioctl (fildes, command, arg)
    int fildes;          /* file descriptor */
    int command;         /* command type */
    struct termio *arg;
```

The commands using this format are:

TCGETA Gets the parameters associated with the terminal and stores them in the **termio** structure referenced by *arg*.

TCSETA Sets the parameters associated with the terminal from the structure referenced by *arg*. The change is immediate.

Note: **TCGETA** and **TCSETA** do *not* get and set a complete record of the state of an **hft** device. See “hft” on page 5-39 for information about high-function terminal devices.

TCSETAF Waits for the output to empty, then flushes the input queue and sets the new parameters.

TCSETAW Waits for the output to empty before setting the new parameters. This form should be used when changing parameters that affect output.

The terminal paging **ioctl** calls have the format:

```
ioctl (fildes, command, arg)
      int fildes;      /* file descriptor */
      int command;     /* command type */
      struct tty_page *arg;
```

The commands using this format are:

TCGLEN Gets the current status of the **tty_page** structure for the terminal specified as **fildes**. If paging is enabled, a value 0x1 is set in **tp_flags**. The **tp_slen** value indicates the screen length in lines.

TCSLEN Sets the status of the **tty_page** structure for this terminal. **tp_slen** means the same here as it does in **TCGLEN**. The **tp_flags** are:

PAGE_SETL	0x4	Set page length using the value in tp_slen .
PAGE_MSK	0x3	Command mask.
PAGE_ON	0x1	Enable paging.
PAGE_OFF	0x2	Disable paging.

Note that the **PAGE_MSK** field is interpreted as an encoding, not as separate flags.

One terminal logging **ioctl** system call has the following format:

```
ioctl (fildes, command, (char *)arg)
      struct tlog *arg;
```

The **tlog** structure is defined in the **sys/termio.h** header file and contains the following members:

```
int    tl_flags
int    tl_msgqid
```


termio

The command using this format is:

TCLOG Requests the terminal logging control functions to execute as indicated by the **tlog** structure. The **tl_flags** are:

TCLOG_ON Determines whether terminal logging is turned on or off.

TCLOG_QID Establishes message queue ID.

If **TCLOG_QID** is set, **tl_msgqid** contains the message queue ID to be used for logging from the terminal. The **tl_msgqid** value must be a message queue identifier returned from a **msgget**.

Additional **ioctl** system calls formats are:

```
ioctl (fildes, command, arg)
    int fildes;      /* file descriptor */
    int command;     /* command type */
    int arg;
```

The commands using this format are:

TCFLSH If *arg* is 0, flush the input queue. A value of 1 indicates flush the output queue. A value of 2 indicates flush both the input and output queues.

TCSBRK Waits for the output to empty. If *arg* is 0, then sends a break (zero bits for 0.25 seconds).

TCXONC Starts or stops control. Suspends output if *arg* is 0. Restarts suspended output if *arg* is a value of 1.

Two query **ioctl** system calls have the following format:

```
ioctl (fildes, command, &arg)
    int arg;          /* returned value */
```

The commands using this format are:

TIOCNOTTY Causes the **tty** controlling this process to disassociate from the process and causes the group ID to clear. Used by background processes to free them from a controlling **tty** and process group.

TIONREAD Gets the summation of the number of characters in the raw and canonical queues.

Two **ioctl** system calls specific to the enhanced edit line discipline have the format:

```
ioctl (fildes, command, arg)
    struct dostmplt *arg;
```

The **dostmplt** structure is defined in the **sys/termio.h** header file, and it contains the following members:

```
char *dt_tbuf  
int   dt_tlen
```

The commands using this format are:

- LDSETDT** Sets the template buffer to contain the first **dt_tlen** characters of **dt_tbuf**, if the enhanced edit line discipline has been entered (if **c_line** equals 1, for example). At most, **DTBSIZE** characters are used. If **dt_tlen** is -1, the template buffer is not initialized.
- LDGETDT** Gets the current contents of the template buffer. The characters in the buffer are written starting at **dt_tbuf**, and **dt_tlen** is set to the number of characters written. At most, **DTBSIZE** characters will be returned. The characters will not be null-terminated.

Two **ioctl** system calls specific to the trusted environment have the format:

```
ioctl (fildes, command, arg)
    int fildes;      /* file descriptor */
    int command;     /* command type */
    int *arg;        /* pointer to an integer */
```

The commands using this format are:

- TCSAK** If the *arg* is defined as 1, then detection of the reserved key sequence **sak** is turned on for the device. A value of 0 turns off **sak** detection. The default is off.
- TCQSAK** If **sak** detection is on, a value of 1 for *arg* is returned. Otherwise, the default 0 (off) is returned.
- TCTRUST** If the *arg* is **TCTRUSTED**, puts the terminal in a trusted state. A value of **TCUNTRUSTED** puts the terminal in an untrusted state.
- TCQTRUST** If the terminal is in a trusted state a value of **TCTRUSTED** will be returned. Otherwise, a value of **TCUNTRUSTED** is returned.

Four **ioctl** system calls specific to security have the format:

```
ioctl (fildes, command, arg);
    int fildes;          /* file descriptor */
    int command;         /* command type */
    int *arg;            /* pointer to an integer */
```

The command using this format are:

- TCSAK** Enables or disables **sak** detection. The arguments are:
- TCSAKON 1** Enables **sak** detection.
- TCSAKOFF 0** Disables **sak** detection.

TCQSAK	Queries whether sak detection is enabled or disabled. The returned argument is either:
TCSAKON 1	sak detection is enabled.
TCSAKOFF 0	sak detection is disabled.
TCTRUST	Makes a terminal trusted or untrusted. The arguments are:
TSTRUSTED 1	Terminal is to be made trusted.
TCUNTRUSTED 0	Terminal is to be made untrusted.
TCQTRUST	Queries whether a terminal is trusted or untrusted. The returned argument is either:
TCTRUSTED 1	Terminal is trusted.
TCUNTRUSTED 0	Terminal is untrusted.

Files

`/dev/tty*`
`/usr/include/sys/ttmap.h`

Related Information

In this book: “`ioctl`” on page 2-407 and “`hft`” on page 5-39.

trace

trace

Purpose

Supports the event-tracing device driver.

Synopsis

```
#include <sys/trace.h>
```

Description

The `/dev/vrmtrace`, `/dev/unixtrace`, and `/dev/appltrace` files are special files that allow event records generated within the VRM, kernel, or application programs to be passed to a user program so that the activity of a driver or other system routines can be monitored for debugging purposes.

The VRM passes buffers of trace entries directly to the driver using unsolicited interrupts.

The **trace** driver supports **open**, **close**, **read**, and **ioctl** system calls. The **ioctl** system call is invoked as follows:

```
#include <sys/trace.h>
ioctl(fildes, cmd, &arg);
int fildes, cmd;

struct tr_struct
{
    unsigned channels; /* enabled channels */
    ushort bufsize;    /* buffer size to use */
    ushort lengths;    /* communication lengths */
    char vmid;         /* VM ID of machine to trace; 0 for current */
    char timer;        /* timer to use, for VRM */
} arg;
```

Valid values of the **cmd** parameter are:

TRCSETC Sets trace parameters. This command instructs the driver to use the parameters provided in structure **arg** to set trace parameters. **bufsize** indicates the size of the buffer to allocate and cannot be changed once it is

set. The **timer** field should always be a value of 0. The **channels** field is a bit map indicating active and inactive channels. As an example, bit 0 corresponds to channel 31, bit 1 corresponds to channel 30, and bit 31 corresponds to channel 0.

TRCGETC Returns the current status of the trace in the structure indicated by **arg**.

The records returned from the trace device are structures with the following format:

```
struct
{
    unsigned stamp;           /* time stamp */
    unsigned short timeext;   /* time stamp extension */
    unsigned short seqno[2];  /* two 16-bit sequence number digits */
    unsigned short hookid;    /* channel no. and trace event code */
    unsigned pid;             /* process-id */
    unsigned short iodn,iocn; /* vrm iodn/iocn or -1 */
    char data [20];           /* more data, depending on code */
};
```

The following subchannels are assigned:

CHANNEL NUMBER

ASSIGNMENT

- | | |
|----|---|
| 22 | Process system calls (acct, alarm, brk, exec, fork, fstat, getgid, getgroups, getpid, getuid, kill, lockf, nice, pause, pipe, plock, profil, ptrace, reboot, setgid, setgroups, setpgrp, setuid, times, ulimit, usrinfo, utssys, wait) |
| 23 | Directory handling system calls (chdir, chroot, link, mknod, unlink) |
| 24 | I/O system calls (access, chmod, chown, close, creat, dup, fcLEAR, fcntl, fsync, ftrunc, ioctl, lseek, open, read, umask, uname, utime, write) |
| 25 | File system system calls (mount, stat, sync, ustat, umount) |
| 26 | Time system calls (stime, time) |
| 27 | Signal system calls (signal, sigblock, sigcleanup, sigpause, sigsetmask, sigstack, sigvec) |

trace

CHANNEL NUMBER	ASSIGNMENT
28	Semaphore system calls (semctl , semget , semop)
29	Message system calls (msgctl , msgget , msgop)
30	Shared memory system calls (shmctl , shmget , shmop)
31	User-defined events

Files

/dev/vrmtrace
/dev/unixtrace
/dev/appltrace

Related Information

In this book: “**trace-on**” on page 2-822, “**trcunix**” on page 2-827, and “**rasconf**” on page 3-189.

The **trace** command in *AIX Operating System Commands Reference*.

The discussion of **trace** in *AIX Operating System Programming Tools and Interfaces*.

The discussion of **trsave** in *Device Driver Development Guide*.

tty

Purpose

Supports the controlling terminal interface.

Synopsis

```
#include <sys/hft.h>  
#include <sys/termio.h>  
#include <sys/tty.h>
```

Description

For each process the **/dev/tty** special file is a synonym for the associated control terminal. This file is useful to programs or shell sequences that want to ensure writing messages on the terminal regardless of how output is redirected. It can also be used for programs that demand the name of a file for output when typed output is desired, and to find out what terminal is currently in use.

Files

```
/dev/tty  
/dev/tty*
```

Related Information

In this book: “hft” on page 5-39.

tty

Chapter 6. Advanced Display Graphics Support Library

About This Chapter

This chapter describes the Advanced Display Graphics Support Library (GSL), an application programming interface to various output devices.

Subroutines, located in the **libgsl.a** library, are provided by the GSL. The **gslerrno.h** header file must be included with an **#include** statement to provide return values for the GSL subroutines.

Note: All GSL parameters are passed by reference, making the subroutines compatible with FORTRAN, in which parameters are always passed by reference. All parameters are therefore passed as pointers in C and are declared as **VAR** parameters in Pascal. The name of a GSL subroutine is always followed by an **_** (underscore) in C and Pascal, but not in FORTRAN.

Note: Beginning with Version 2.2 of the GSL, applications can be linked to either a shared or an unshared version of the GSL library. Before writing an application program that uses the GSL, see "Using the GSL Libraries" on page 6-23.

The **/usr/lib/samples** directory contains routines that provide clipping and transformation functions on a normalized device coordinate (NDC) system, as well as routines that provide additional functions. The **README.gslex** file in this directory provides more information.

The Extended and Enhanced GSL subroutines that reside in the **/usr/lib/samples directory are merely examples, provided for the sole purpose of illustrating that the basic GSL subroutines can be used to create extended or enhanced subroutines. The extended and enhanced GSL subroutines are each provided "as is" without warranty of any kind, either express or implied, including but not limited to the implied warranties of merchantability and fitness for a particular purpose. The entire risk as to the quality and performance of each of the extended or enhanced GSL subroutines is with you.**

Overview

This overview section contains information to help you become acquainted with the terms, major concepts, and functionality of the GSL. The first section lists the contents of the chapter, followed by a section that defines special GSL terms. The next segment contains descriptions of the functionality provided by the GSL and an explanation of some of the important concepts needed to use the GSL. Later sections describe the attributes, cursor operations, and coordinate clipping and transformation capabilities of the GSL. Refer to the following list of contents for the first page of each of these discussions.

Contents

The following list can be used to locate information on specific topics relating to the GSL:

About This Chapter	6-2
Overview	6-2
Contents	6-3
Definitions	6-4
Concepts	6-5
Attributes	6-7
Common Attributes	6-7
Unique Attributes	6-8
Cursor Operations	6-12
Coordinate Clipping and Transformation	6-13
Functional Categories of Subroutines	6-14
Control	6-14
Output	6-15
Service	6-17
Pixel Block Transfer	6-18
Cursor	6-18
Attribute	6-19
Input	6-19
Query	6-20
Writing GSL Application Programs	6-20
Displays	6-21
Printers and Plotters	6-21
Using the GSL Libraries	6-23
Notes on the lpp.linkgsl Shell Script	6-24
Example lpp.linkgsl Shell Script	6-25

Definitions

The following terms are defined for this chapter:

Frame buffer	A display adapter frame buffer is memory storage containing a representation of a display image.
Geometric text	The two types of text supported by the GSL are annotated, or standard, text and geometric text, which is also referred to as a programmable character set (PCS) or stroke text. For more information on annotated text, see “fonts” on page 3-79.
KSR mode	KSR (keyboard send-receive) mode causes a virtual terminal to act like a standard ASCII terminal, with some RT extensions, for both input and output. See “hft” on page 5-39 for more information about KSR mode.
Monitor mode	In monitor mode, a virtual terminal lets an application directly access the display adapter without conflict with the standard virtual terminal output mechanism. Further information about monitor mode is found under “hft” on page 5-39.
Pick device	Valid only for the IBM 5081 Display Adapter, a pick device is a hardware-assisted event. An area of the screen around the active cursor is specified and pick is enabled. The adapter resets a counter to zero, then counts each graphics primitive command issued. This count identifies each output function. If a graphics primitive then intersects the area defined around the cursor, the IBM 5081 Display Adapter returns the count associated with the primitive to the system. This information appears on the GSL input ring as a pick event , and helps an application determine what an operator is selecting on the screen.
Pixel	A pixel, or picture element, is one point in the frame buffer or on the display.
Pixel map	Also known as a pixmap , this is an object that defines the characteristics of a rectangle. See “gsxblt” on page 6-164 for a list of the elements defined by a pixel map.
Ring Buffer	A virtual terminal in monitor mode can share a ring buffer with an application and place data from input devices in the buffer. The ring buffer mechanism dramatically shortens the input data path from the virtual terminal to the application.

Concepts

The GSL allows applications to perform graphics operations without the need to directly manipulate the underlying hardware. The GSL also supports the display of fixed-spaced characters in text.

The GSL assumes that an application using it runs in its own virtual terminal. A virtual terminal can operate in either KSR mode (the default) or in monitor mode. An application can use monitor mode and the ring buffer to derive its own graphics interface. The GSL provides an interface that lets a user generate graphics interactively without detailed knowledge of the display adapter and input data formats. The GSL works only with the application virtual terminal in monitor mode. Part of the GSL initialization is to place the virtual terminal in monitor mode. This forces some restrictions on the use of the display adapter. The application virtual terminal can be one of several virtual terminals opened by a user, but only one virtual terminal can be *active* for input at any time. Several virtual terminals can be active for output at any time if multiple displays are attached, with one virtual terminal active for output on each display. All virtual terminals but one, however, are *inactive* for input at a given time. The active virtual terminal for input can write to the display adapter and can receive input from devices.

Applications must respond to user requests to become active or to release control of the display (become inactive). The transfer of control of the display occurs with two signals (a release request, **SIGRETRACT**, and a grant notification, **SIGGRANT**) and a write to the **hft** device driver to acknowledge the release signal. After initialization, the GSL processes these two signals and writes to the device driver so that it can determine when it can and cannot write to the adapter. Routines that an application supplies that get called by the GSL signal handlers can be identified by the application during GSL initialization. The application can therefore respond appropriately to requests to be active or inactive.

The GSL provides a set of graphics output functions. Applications can supply additional functions that access the display adapter directly. Such an application routine can function only when the virtual terminal is active, and the virtual terminal must not become inactive while the routine is operating. The GSL provides a function that indicates to the application whether its virtual terminal is active or inactive, and if active, postpones GSL processing of the **SIGRETRACT** signal until the application has finished modifying the display. Another function causes the GSL to resume processing of the signal.

One of the GSL output functions or an application-supplied output function can be operating at the time of the **SIGRETRACT** signal. If this is the case, the function only has 30 seconds (real time) to complete the adapter operation and acknowledge the **SIGGRANT** after receiving that signal. After the 30 seconds, the **hft** device driver sends a **SIGKILL** signal that terminates the virtual terminal. The application should be designed with this consideration in mind, or the user should be made aware of the time limit for applications that involve switching virtual terminals and have lengthy drawing operations.

The virtual terminal subsystem dictates that when a monitor mode virtual terminal becomes inactive and then active, the application must restore the display adapter state.

At initialization the application can direct the GSL to use either of two mechanisms for restoration.

GSL Control

The GSL saves the frame buffer at the time of the **SIGRETRACT** and restores it and the appropriate adapter state, such as the color map, at the time of the **SIGGRANT**. Unfortunately, saving and restoring large frame buffers can be relatively expensive in terms of time and virtual storage space. Under this mechanism, an output operation initiated while the application virtual terminal is inactive suspends the application until its virtual terminal becomes active. If the virtual terminal is inactive when the application requests postponement of **SIGRETRACT** signal handling, the GSL suspends the application until the virtual terminal becomes active.

Application Control

The GSL saves the adapter state at the time of the **SIGRETRACT** request and calls an application routine (if provided) at the time of the **SIGGRANT**. This routine could process the applications data structure(s) to reconstruct the display adapter state. Under this mechanism, an output operation initiated while the application virtual terminal is inactive causes the output routine to return without writing to the display adapter. The routine returns a code indicating an invalid status in this circumstance. If the virtual terminal is inactive when the application requests postponement of **SIGRETRACT** signal handling, the GSL sends a code indicating that the application cannot access the display.

Regardless of the mechanism chosen, the GSL calls an application routine (if provided) at the time of the **SIGRETRACT** request and calls an application routine (if provided) at the time of the **SIGGRANT** notification. One or both restoration routines can be chosen for an application as appropriate.

An application cannot write to standard output (using system write) on a virtual terminal that is in monitor mode. However, at initialization, the GSL accepts a specified file descriptor as the monitor mode virtual terminal from the application, and directs output to this file descriptor. An application can use more than one virtual terminal, and the virtual terminals can be mapped to different displays simultaneously. This reserves standard output for other uses such as **sdb**, the symbolic debugger.

When the ring buffer mechanism is used for processing input, the virtual terminal places input from the keyboard, locator, LPFK, valuator, or pick device in a ring buffer shared between the application and the virtual terminal. The virtual terminal causes the generation of the **SIGMSG** signal when it places the data for an input event in an empty ring buffer. At initialization, an application can select either method. However, the GSL supports only the ring buffer mechanism to optimize performance. If used, a ring buffer must be allocated by the application and made available to the GSL at initialization. The GSL sets up the virtual terminal linkage to the buffer and sets up a signal handler to catch the **SIGMSG** signal that it uses to satisfy application requests for input.

The application must then let the GSL process the ring buffer input pointer and parse the input events by invoking the appropriate input function. Whenever the application has

selected the ring buffer mechanism, the application can use GSL input to enable and disable input events.

The application can provide a signal handler to catch the **SIGMSG** signal if all of the following conditions are met:

1. The signal handler is set up after the GSL is initialized.
2. The signal handler is set up using the **SIGVEC** enhanced signal function. **SIGVEC** returns the address of the GSL signal handler.
3. The signal handler must indirectly call the GSL signal handler before doing anything else. The indirect call uses the address returned by the **SIGVEC** signal.

Enhanced signals are used to block further reporting of the signal being processed until the signal handler returns. When the signal handler returns, the signal is automatically reset and unblocked.

When keyboard events are enabled, the virtual terminal puts *all* keystrokes in the ring buffer, including those that can normally have special meaning to the operating system (such as break). The application can let the system continue processing certain keystrokes by setting the virtual terminal break map.

For further information on monitor mode operation, see the discussion of the virtual terminal subsystem in *VRM Programming Support*.

Attributes

A set of **attributes** that determine how a function works, or determine appearance characteristics on a display, govern all GSL operations affecting the frame buffer. Attributes are characteristics that do not change often and therefore do not need to be parameters for the output functions. Some common attributes govern all output operations while others are unique to a particular category of output.

Common Attributes

Color display adapters can be considered to have multiple storage planes or layers forming the frame buffer, with each plane acting like the single frame buffer for a bilevel monochrome display. When writing a pixel into a multiplane frame buffer, you can write to all the planes or to a subset. The GSL **plane mask** attribute identifies which planes of the frame buffer GSL functions modify.

The color of a pixel on the display is ultimately determined by the color value of the pixel stored in the frame buffer. There are **VLT-based adapters**, in which the pixel color value serves as an index into a video lookup table (VLT). The entry in the VLT for an index contains a value for each of the red, green, and blue digital-to-analog converters (DACs) on the adapter, which drive the color guns in the display tube. The actual color resulting from a particular pixel color value (VLT index) depends on the values loaded into the VLT,

which can be any values. There are also **true color adapters** in which the pixel color value actually drives the DACs, without the level of indirection forced by the VLT.

An application can determine the mapping from the color used in operations on the frame buffer to the actual color shown on a display by using the GSL **color map** attribute. For VLT-based adapters, the GSL actually loads the adapter VLT, using color values provided by the application; the **color** used by the application is really an index into the VLT. For true color adapters, the color map serves strictly as an internal mapping from the color value specified to the actual color value loaded into the frame buffer, and the **color** used by the application is an index into the mapping table.

The application can set the color map by providing an array of color specifications. The maximum number of specifications is dependent on the display adapter and is determined by the number of VLT entries, or by the number of bit planes for true color adapters. The color specification for each color index comprises three intensity values, one each for the red, green, and blue DACs. Each intensity value must range from 0 to 0x3FFF. For a VLT-based display adapter, the GSL maps the color specification to the nearest available color produced by the adapter; the GSL truncates the intensity value for a color to produce a value equal in resolution to the DAC for that color.

The **logical operation** attribute determines how the GSL combines the pixels it generates with the current contents of the frame buffer. Sixteen Boolean combinations exist between a source (the GSL-produced pixels) and a destination frame buffer, but can only be used with the IBM 5081 Display Adapter. The GSL does, however, ensure support for the most recognizable and useful Boolean combinations (replace and exclusive-or) regardless of hardware support.

The following table shows the categories of functions to which the common attributes apply.

Area	Output	Pixel Block	Cursor
Plane mask	yes	yes	yes
Color map	yes	yes	yes
Logical operation	yes	no	no

Unique Attributes

Some unique attributes, along with the plane mask, color map, and logical operation attributes, govern the GSL functions that affect the frame buffer.

Lines

The GSL draws all lines a single pixel thick. The following unique attributes that govern line drawing can be changed:

<i>Line style</i>	Determines the pattern appearance of the line. The line style attribute provides for solid, dashed, dotted, dashed-dotted, and dashed-dotted-dotted lines, and for line patterns defined by the application.
--------------------------	--

<i>Line color</i>	Is an index into the color map table (or VLT).
--------------------------	--

Markers

The marker attributes determine characteristics of symbols used to mark points. The GSL provides a set of predefined markers for the application to select. A marker can be custom-defined by the application.

The application can change the following unique attributes that govern marker operations:

<i>Marker color</i>	Sets the color of a marker, and is an index into the color map table (or VLT)
----------------------------	---

<i>Marker origin</i>	Sets the point in the marker pattern that is placed at the position indicated by the application for the polymarker subroutine
-----------------------------	--

<i>Marker style</i>	Selects predefined or custom markers
----------------------------	--------------------------------------

<i>Marker width</i>	Defines the width of the pattern for a custom marker
----------------------------	--

<i>Marker height</i>	Defines the height of the pattern for a custom marker
-----------------------------	---

<i>Marker pattern</i>	Sets the form of the custom marker, and is a bit array defined by the application.
------------------------------	--

Text

The GSL places characters with a transparent background. That is, only the strokes in a character change data in the frame buffer. These unique attributes govern text operations and can be set by an application:

<i>Text font</i>	Sets which of the available fonts is used for the characters
-------------------------	--

<i>Text color</i>	Sets color and brightness of the text and is an index into the color map table or VLT
--------------------------	---

<i>Code page</i>	Sets the page from which graphic symbols are drawn
-------------------------	--

<i>Baseline direction</i>	Sets the direction in which characters are written to the baseline for the text. The baseline for the string is placed at the location given in the command to write text.
----------------------------------	--

Xtext

The GSL provides a high-performance text drawing routine. This routine draws both the foreground and background for each character. In addition, clipping and logical operations are also provided. Note that, unlike the standard GSL annotated text, xtext does not provide the baseline direction attribute. Xtext is always drawn from left to right.

xtext foreground color

Sets the color of the foreground (or 1 bits) of the text and is an index into the color map table or VLT.

xtext background color

Sets the color of the background (or 0 bits) of the text and is an index in the color map table or VLT.

xtext logical operation

Determines how the GSL combines the pixels of the text with the current contents of the frame buffer.

xtext clip box

Specifies a rectangular area of pixels in the frame. The **gsxtxt** subroutine will not place full or partial characters outside this rectangular area.

Filled Areas

The edges of an area are treated as part of the area and only define the area to be filled. The GSL does not treat the edges of an area as lines. The application can change the following unique attributes that govern fill operations:

Fill color

Provides an index into the color map table (or VLT)

Fill pattern

Specifies the identifier for the pattern used to fill the area.

Cursor

The GSL provides a single cursor for the application at any given time. The application can use either a single-color cursor or a multicolor, masked cursor. The single-color cursor provides higher performance and allows a larger cursor pattern. The multicolor cursor provides foreground and background colors, clipping, logical operations, and masking.

The application can change the unique attributes that govern cursor operations. For the single-color cursor, these attributes are:

Cursor pattern

Sets the cursor shape and consists of a bit array defined by the application. The minimum and maximum sizes for the cursor pattern are device-dependent and available to the application.

Cursor color

Sets the cursor color and provides an index into the color map table or VLT.

Cursor origin

Sets the point in the cursor pattern that is placed at the position indicated by the application during cursor movement.

For the multicolor cursor, these attributes are:

Multicolor cursor pattern

Sets the cursor shape and consists of a bit array defined by the application. The minimum size is 1 and the maximum size is 32 for both width and height.

Multicolor cursor foreground color

Sets the cursor foreground color and provides an index into the VLT.

Multicolor cursor background color

Sets the cursor background color and provides an index into the VLT.

Multicolor cursor origin

Sets the point in the cursor pattern that is placed at the position indicated by the application during cursor movement.

Multicolor cursor mask

Sets the mask to be applied to the multicolor cursor pattern, using a bit array defined by the application. The mask size must match the size of the multicolor cursor pattern.

Multicolor cursor logical operation

Defines how the GSL combines the pixels of the multicolor cursor with the current contents of the frame buffer.

At GSL initialization, some of the attributes receive default values. The attributes and their default values are listed in the following table:

Attribute	Default Value
Color map	Device-dependent
Plane mask	All planes enabled
Logical operation	3 (replace)
Line style	Solid
Line color (index)	7 (white)
Font	Device-dependent
Code page	P0
Baseline direction	0 (left to right)
Text color (index)	7 (white)
Fill color (index)	7 (white)
Fill pattern	0 (solid)
Cursor pattern	Undefined

Figure 6-1 (Part 1 of 2). Default Attribute Values

Attribute	Default Value
Cursor color	Undefined
Cursor origin	Undefined
Multicolor cursor pattern	Undefined
Multicolor cursor foreground color	Undefined
Multicolor cursor background color	Undefined
Multicolor cursor origin	Undefined
Multicolor cursor mask	Undefined
Multicolor cursor logical operation	Undefined
Xtext foreground color	Undefined
Xtext background color	Undefined
Xtext logical operation	Undefined
Xtext clip box	Undefined

Figure 6-1 (Part 2 of 2). Default Attribute Values

Cursor Operations

The GSL provides one cursor for applications. The GSL cursor is nondestructive; the contents of the display adapter frame buffer remain intact when the cursor is already visible and subsequently moved or made invisible. This is achieved in a device-dependent manner. The GSL uses any hardware cursor support available.

The application totally controls the placement and visibility of the cursor. The GSL input functions do not provide cursor movement, and the GSL output routines do not check whether they are drawing over the cursor and do not automatically erase and restore the cursor or check for interference. It is therefore possible for the output routines to overwrite the cursor. When the cursor is moved on a display without hardware cursor support, any primitives that overwrite the cursor will themselves be overwritten when the area at the previous cursor position is restored. The application should erase and redraw the cursor as appropriate to avoid conflict.

The GSL provides two styles of cursor. One style, referred to as the single-color cursor, provides high performance and device-dependent, but larger, cursor pattern sizes. The second style, called the multicolor cursor, is a two-color, masked, clipped cursor that has both the replace and the exclusive-or logical operations. Only one style of cursor can be defined and used at any one time.

Coordinate Clipping and Transformation

For simplicity and optimized performance, the base GSL does not perform general clipping or transformation on coordinates. Most of the output functions accept coordinates in the first quadrant (0,0 is the lower left corner) and convert them as necessary to the target quadrant required for the frame buffer of the specific device.

The coordinate system is device-dependent, and any point outside the frame buffer range can result in a write to any address on the I/O bus. For this reason, the GSL checks coordinates sent as parameters and also coordinates generated internally against the frame buffer boundaries.

Note: Except for the **gsxtat** and **gsxtxt** routines, any coordinate outside the frame buffer is invalid and produces an invalid status return. (See the following explanation of **xtext**.)

The display results depend on the function invoked. Coordinates outside the frame buffer are handled as follows:

- | | |
|---------------------|---|
| Lines | The GSL checks the input coordinates as it draws the lines. Thus, for polylines and multilines, the part of the sequence up to the invalid coordinate results in lines on the display. The functions return an error code upon encountering an invalid coordinate, drawing no further lines. |
| Text | If the start point of the text string is invalid, the GSL returns immediately with an error code. Invalid internal coordinates can be generated if the start point is valid but part of the text string overflows the frame buffer. If the application places the baseline such that the characters must be clipped vertically (for example, the top half of the string is out of the frame buffer), the GSL writes none of the characters in the string. If the application places the baseline such that a character in the string overflows the frame buffer, that entire character and the rest of the string is truncated. |
| Xtext | The GSL provides clipping of this text style through the xtext clip box attribute. Coordinates outside the frame buffer are valid and will be clipped. |
| Filled Areas | The GSL checks the coordinates of filled areas before it writes to the frame buffer. It returns an invalid return code for any invalid coordinate found before writing to the frame buffer. |
| Cursor | The application can place the cursor origin anywhere within the cursor pattern. If the single-color cursor origin is placed so that any part of the cursor falls outside of the frame buffer, an error code is returned and the cursor is not moved. For the multicolor cursor, the application can place the origin anywhere within the cursor pattern. If the multicolor cursor origin is placed so that |

any part of the cursor falls outside the frame buffer, then the part outside the frame buffer is clipped. If the origin of the multicolor cursor is placed outside the frame buffer, then an error code is returned and the cursor is not moved.

Pixel Block Transfer If any portion of the source or destination rectangle lies outside its pixmap, the GSL returns immediately with an error code.

Functional Categories of Subroutines

The base GSL is device-dependent in that it provides some functions for the IBM 5081 Display that it does not provide for other displays. It also does not scale, clip, or transform coordinates for any display. Coordinates passed to the base GSL functions are therefore device-dependent, and limited to the device boundaries. The GSL does make device-specific information available through query commands, letting an application perform appropriate clipping and transformation. It also indicates the logical operations supported by the hardware.

The base GSL is organized into several major function areas:

- Control
- Output
- Service
- Pixel Block Transfer
- Cursor
- Attribute
- Input
- Query.

The following sections provide an overview of the functions in each area.

Control

The base GSL provides functions to initialize and terminate the GSL and to coordinate direct application access to the display device. At initialization, the GSL sets up its required environment and establishes **monitor mode** on the application virtual terminal. Monitor mode provides the GSL with direct access to the display adapter without interference from the virtual terminal subsystem. Monitor mode operation also gives faster access to input event information. At termination, the GSL cleans up after itself and returns the application virtual terminal to KSR mode.

These subroutines perform overall control operations of the GSL environment.

gsinit	Initializes the GSL subroutines, establishes monitor mode on the application virtual terminal, and allows specification of application-supplied signal processing routines.
gslock	Locks the virtual terminal so that the application can access the display adapter directly.
gsterm	Terminates the GSL, returns the application virtual terminal to KSR mode.
gsunlk	Unlocks the virtual terminal, returning control to the GSL.

Output

The GSL output functions provide an application with capabilities to perform graphics operations on output devices. The output functions can be divided into these categories:

Drawing lines

The GSL provides functions to draw:

- A line between two points
- A series of lines connecting a sequence of points
- A series of lines connecting alternate pairs in a sequence of points.

GSL lines are a single pixel thick. Specific attributes allow lines of different colors and patterns.

Drawing polymarkers

The polymarker subroutine in the GSL lets a defined marker be drawn for a sequence of points. The definition of the marker includes specific attributes such as color, style, width, height, pattern, and origin. The pattern attribute is a raster image to be used as a marker. The origin attribute controls the placement of the polymarker pattern at the points specified by the polymarker subroutine.

Writing annotated text

The GSL provides a function to write a text string to the display adapter at a given starting position. Character placement is with a transparent background so that the GSL changes only the character shape (foreground), not the entire character box. Specific attributes allow text in different fonts, colors, code pages, and directions.

In addition, the GSL provides a text drawing facility that changes both the foreground and background of the frame buffer where the character box is placed. This facility, called **xtext**, allows text in different fonts, foreground and background colors, and logical operations.

Writing geometric text

The GSL provides functions to write geometric text strings.

Drawing curves

The GSL provides functions to draw circles, arcs, and ellipses. These functions are used to achieve the best performance and quality possible so that your programs can realize the full capability of your display.

Filling areas

The GSL provides functions to draw filled rectangles and general polygons, circles, and ellipses. In addition, the GSL allows curves to be combined with polylines to fill complex shapes. See “gsbply” on page 6-28, “gseply” on page 6-61, and “gspcls” on page 6-125. These functions allow applications to use the higher performance possible with rectangles. The GSL also provides a **color zero** function to clear the display to the background color. Specific attributes allow different colors and patterns. See “Attributes” on page 6-7.

Adapter level interface

The GSL provides a function to send adapter level commands directly to the display adapter hardware. See “gssend” on page 6-149 for more information.

Each category of the output functions is governed by a set of attributes. Some attributes determine characteristics that are specific to the category, such as color or pattern. These attributes are common to all categories:

color table

Maps color names or values to the actual color on the display (see “Common Attributes” on page 6-7).

plane mask

Determines which of the display adapter storage planes are modified by the output functions.

logical operation

Determines how the GSL combines the foreground or background color for each pixel that is produced by a primitive with the current color of the destination pixel in the frame refresh buffer.

These output subroutines write to the display adapter frame buffer, generally producing output on a display screen:

gsbply	Begins a polygon.
gscarc	Draws a circular arc of a specified radius between two points.
gscir	Draws a circle.
gscrls	Clears the display screen, filling it with the background color.
gscrca	Draws a circular arc between two angles.
gsdjply	Draws a disjoint polyline, or a path of straight lines that are not connected.
gseara	Draws an elliptical arc between two angles.
gsearc	Draws an elliptical arc of specified axes and angle between two points.
gsell	Draws an ellipse.
gseply	Ends a polygon.
gsfci	Fills a circle.

gsfell	Fills an ellipse.
gsfrec	Draws a filled rectangle.
gsfply	Draws a filled polygon.
gsgtxt	Displays a geometric text string, with NDC transformations supported.
gsline	Draws a line between two points.
gsmult	Draws a multiline, or a set of straight lines that connect alternate pairs of points in a sequence.
gspcls	Closes a polygon.
gsplym	Draws a polymarker, a marker (such as a dot or plus sign) at each of a specified set of points.
gspoly	Draws a polyline, or a path of straight lines that connect a sequence of points.
gssend	Sends adapter level commands directly to the display adapter hardware.
gsstxt	Displays a text string.
gsxtxt	Displays a text string in the rtfont format.

Service

The GSL provides functions for defining a circular or elliptical arc. These functions convert circular or elliptical arc definitions into sets of vertices. The resulting set of vertices can be drawn, using the **gsline** subroutine, or combined with other polylines to draw or fill more complex shapes.

The attributes that can be used for drawing lines or filling areas apply here, including style, color, logical operation, pattern, and others.

Arcs are specified by beginning and ending points or beginning and ending angles, and follow the counterclockwise direction. If the beginning and ending points are identical, then the list of vertices corresponding to a full circle or ellipse is returned. This allows circles or ellipses to be treated as a special case of closed arcs. If off-axis, ellipsis angle is specified in degrees. There are four levels of precision for the conversion of an arc into a set of line segments.

These subroutines facilitate the drawing of circular and elliptical arcs.

gscenv	Converts a circle to a set of vertices (polyline).
gsecnv	Converts an ellipse to a set of vertices (polyline).

Pixel Block Transfer

The GSL provides functions to move a rectangular block of pixels from either the display adapter frame buffer or storage to either the display adapter or storage. If the source rectangle or destination rectangle reside in a color display adapter frame buffer, this operation is affected by the plane mask attribute. If the destination rectangle resides in a color display adapter frame buffer, this operation is affected by the color map attribute.

These subroutines allow a program to:

- Save a block of pixels from the frame buffer.
- Restore a block of pixels from the frame buffer.
- Move a rectangular shape from adapter memory to pixel memory.
- Move a rectangular shape from adapter memory to system memory.
- Move a rectangular shape from one area in system memory to another.
- Move a rectangular shape from one area of adapter memory to another.
- Move a tile rectangle to any area of visible pixel memory.

gsrrst	Restores a rectangular block.
gsrsav	Saves a rectangular block.
gsxblt	Moves a rectangular block from one location in memory or display adapter frame buffer to another location in memory or display adapter frame buffer.
gsxcnv	Converts pixel format data to and from plane format data.
gsxptr	Handles FORTRAN addressing of pixel map data.

Cursor

The GSL provides functions to draw and undraw a nondestructive cursor. The application is responsible for the placement and visibility of the cursor. The input functions do not provide for cursor movement, nor do output or pixel block transfer functions check whether they interfere with the cursor. Anything unintentionally placed over the cursor is modified when the cursor moves. The color map and plane mask attributes govern the cursor functions. Cursor pattern and color can be defined by attributes.

These are the cursor subroutines:

gsecur	Erases the cursor and makes it invisible.
gsmcur	Moves the cursor and makes it visible.

Attribute

The GSL provides functions to set the global attributes and all of the output category specific attributes. The GSL also provides functions to set attributes of some of the input devices.

These subroutines set attributes for both input and output operations:

gscatt	Sets the single-color cursor attributes.
gscmap	Sets the color map.
gsfatt	Sets the fill attributes.
gsztat	Sets the attributes for the geometric text drawing operation, gsgettext .
gslatt	Sets the line attributes.
gslcat	Sets the locator attributes.
gslpat	Sets the LPFK indicators.
gslop	Sets the logical operation used for drawing lines.
gsmask	Sets the plane mask.
gsmatt	Sets the attributes for the polymarker operation, gsplym .
gsmcat	Sets the multicolor cursor attributes.
gsmfld	Loads multiple fonts simultaneously.
gspp	Sets plotter pen speed as a percentage of the plotter maximum speed.
gstatt	Sets the attributes for the text output operation, gstext .
gsulns	Sets the user line pattern.
gsvgrn	Sets the valuator granularity.
gsxtat	Sets the attributes for drawing annotated text in the rtfon format, using the gsxtxt routine.

Input

An application using the GSL can receive input with the standard **read** system call or through a faster mechanism available through the virtual terminal. While the GSL allows an application to use the standard mechanism, it provides no input support for it.

The GSL accepts input from several sources:

- Keyboard
- Locator, which can be a mouse or a tablet
- Lighted programmed function keys (LPFKs)
- Valuator dials
- Pick device (valid for IBM 5081 Display Adapter only).

Input from these devices is viewed as discrete events, with input data associated with each event.

The GSL provides subroutines to enable or disable input from any device, and a subroutine that lets a program suspend execution until one of the enabled events occurs. The latter

subroutine also parses the raw data generated by the virtual terminal and makes the parsed information available to the application. In addition, two GSL subroutines allow you to enable or disable pick events.

The state established (enabled or disabled) remains in effect when the GSL terminates. Note that for these subroutines to work properly, a valid input ring buffer must have been specified to the **gsinit** subroutine.

gsdpik	Disables picking.
gsepik	Enables picking.
gsevds	Disables the reporting of input events.
gseven	Enables the reporting of input events from the keyboard, locator, LPFK, or valuator.
gsevwt	Waits for an input event and parses the raw data.

Query

The GSL provides functions for applications to query the active display adapter characteristics, the currently active annotated or geometric text font, and some input device characteristics. Through query functions, an application can derive the information necessary to deal with any device dependencies. Note that **gsinit** must be invoked before calling any of the query subroutines.

These are subroutines that provide query functions:

gsqdsp	Returns device-specific information about the display adapter and display monitor.
gsqfnt	Returns information about the current annotated text font.
gsqgtx	Returns information about the current geometric text font.
gsqloc	Returns device-specific information about the locator.

Writing GSL Application Programs

The following sections contain information that will help you take advantage of the capabilities of the GSL when writing application programs. “Displays” describes the display support provided by the GSL; “Printers and Plotters” explains available printer and plotter support; and “Using the GSL Libraries” contains instructions on using libraries and setting up the installation of your application program.

Displays

The GSL provides support for the following all-points-addressable display adapters and displays:

6153 Display

A monochrome 720 by 512 adapter; a 12-inch display.

6154 Display

A 16 of 64 color, 720 by 512 adapter; a 14-inch display.

6155 Display

A monochrome 1024 by 768 adapter with significant graphics assist; a 15-inch display.

5081 Display

A 256 of 4096 color, 1024 by 1024 adapter or a 256 of 16M color, 1024 by 1024 adapter; a 16-inch or 19-inch display.

Support for additional displays and adapters can be added to the GSL. For instructions on how to add this support, see *Device Driver Development Guide*.

The GSL automatically uses the correct configuration for an installed display adapter at initialization. It accepts input from any device that conforms to the virtual terminal interface as described in *VRM Device Support*. The GSL supports one or more of the following input devices in an application:

- Keyboard
- Mouse or tablet
- Lighted Program Function Keyboard (LPFK)
- Valuator
- Pick device (valid for IBM 5081 Display Adapter only).

At least one input device is always available; the virtual terminal subsystem determines that, at minimum, keyboard input is accepted.

Printers and Plotters

Note: The printer text data stream supports only the ASCII character set.

Before an application can use GSL subroutines to generate graphic output to a printer or plotter, the Graphics Development Toolkit device drivers must be installed on the system. See the section about installing additional operating system programs in *Installing and Customizing the AIX Operating System* for instructions on how to do this.

Certain information about the device must be defined using AIX environment variables. If you enter the definitions at the shell command line, then they remain in effect only for the current login session. If you want these definitions to remain in effect for future login sessions, add them to the **.profile** file in your home directory. To define this information

permanently for all users, add it to the `/etc/profile` file. See the **sh** command in *AIX Operating System Commands Reference* for more information about AIX environment variables, which are also called **shell variables**.

1. Define the path to the Graphics Development Toolkit device drivers:

```
VDIPATH = /usr/lpp/vdi/drivers  
export VDIPATH
```

2. Define a logical identifier for the device as an environment variable, and set its value to indicate the type of printer or plotter device:

```
devname = vdixxxx  
export devname
```

The name you use in place of *devname* can be any sequence of up to 11 alphanumeric characters. This is the name that you specify in the *fildevs* parameter of the **gsinit** subroutine.

The value of the environment variable, **vdixxxx**, is one of the following names:

vdix3812	IBM 3812 Printer
vdix4201	IBM 4201 Printer
vdix5152	IBM 5152 Printer
vdix5182	IBM 5182 Printer
vdix6180	IBM 6180 Plotter
vdix7371	IBM 7371 Plotter
vdix7372	IBM 7372 Plotter
vdix7375	IBM 7374, 7374-1, or 7375-2 Plotter

3. You can specify a printer or plotter as **vdixxxx** in step 2; **vdixxxx** must be associated with an AIX special file:

```
vdixxxx = /dev/yyyy  
export vdixxxx
```

If you do not need output from a specific printer device, you can pipe the output to a printer queue:

```
vdixxxx = "| print -plot [queue]"  
export vdixxxx
```

Note: You can only pipe output to a queue for printer devices, not for plotters.

4. If you are using an IBM 3812 Pageprinter, then you should set and **export** (as in the examples below) the following additional environment variables:

MARGIN	Set to either TRUE or FALSE , indicating whether to leave 1/4-inch margins. If not defined, the default is MARGIN = FALSE .
---------------	--

ORIENTATION Set to either **LANDSCAPE** or **PORTRAIT**, indicating horizontal or vertical page orientation, respectively. ***Landscape orientation*** rotates the image 90 degrees so that the horizontal axis of the image goes down the length of the page. If not defined, the default is **ORIENTATION=PORTRAIT**.

Examples

- The following example defines GRAPHDEV as the logical name of an IBM 3812 printer that is configured as **/dev/tty1**. This configuration specifies ***portrait orientation*** (output frame vertical dimensions are greater than horizontal) and no margins.

```
VDIPATH=/usr/lpp/vdi/drivers
GRAPHDEV=vdi3812
vdi3812=/dev/tty1
MARGIN=FALSE
ORIENTATION=PORTRAIT
export VDIPATH GRAPHDEV vdi3812 MARGIN ORIENTATION
```

- The next example defines GRAPHDEV as the logical name of an IBM 3812 printer that is already configured as the device that serves printer queue **lp0**. This configuration specifies ***landscape orientation*** (output frame horizontal dimensions are greater than vertical) and 1/4-inch margins.

```
VDIPATH=/usr/lpp/vdi/drivers
GRAPHDEV=vdi3812
vdi3812="| print -plot lp0"
MARGIN=TRUE
ORIENTATION=LANDSCAPE
export VDIPATH GRAPHDEV vdi3812 MARGIN ORIENTATION
```

Note: Only the output subroutines, listed in "Output" on page 6-15, are valid for printers and plotters.

Using the GSL Libraries

Three versions of the GSL subroutine library are provided in **/usr/lib**:

- An unshared library, named **libgsl.a**.
- A shared library, named **libogsl.a**, and its corresponding text image file, **ogsl.txt**.
- An unshared library, named **libxgsl.a**.

IBM strongly recommends that you use the shared GSL library, since doing so can save virtual memory and disk storage space. For more information about shared libraries, see “Shared Libraries” in *AIX Operating System Programming Tools and Interfaces* and the **shlib** command in *AIX Operating System Commands Reference*.

A GSL application should be relinked any time that the GSL libraries are updated. If, for example, support for a new type of display is added to the libraries and the application is not relinked, then the application cannot access the new display and can produce unpredictable results.

You can relink GSL applications yourself, but the libraries can be updated without your knowledge. Therefore, IBM recommends a facility that automatically relinks all GSL applications when the libraries are changed. To use this facility, your application should follow these conventions:

- Be provided as object files, *not* as prelinked executables.
- Provide an **sh** shell script named **lpp.linkgsl** that links the object files into the final executable form.

The **lpp.linkgsl** shell script must be located in a directory named **/usr/lpp/pgm-name**, where *pgm-name* is a name that uniquely identifies your application program. Whenever the GSL libraries are changed, such as for the addition of new display support, each of the **/usr/lpp/pgm-name** directories is searched for an **lpp.linkgsl** file. Each **lpp.linkgsl** is executed, relinking the GSL applications with the updated subroutine library.

If your GSL applications do not adhere to the conventions that provide for automatic relinking of your applications whenever the GSL libraries are updated, and your applications are linked with the shared version of the GSL library, then your applications will not execute correctly once the update has been applied. In this case, you will have to relink your GSL applications manually. If your GSL applications are linked with the nonshared GSL library, they will continue to execute correctly, but they will not access the updated GSL library.

Notes on the **lpp.linkgsl** Shell Script

1. If you are writing an application program to be installed on more than one system, IBM strongly recommends that you package your application to be installed from diskette with the standard AIX conventions of the **installp** command. Your **install** procedure should run **lpp.linkgsl** to link the application with the GSL initially. You can find instructions for creating a diskette in the proper format under “Installing and Updating a Program” in *AIX Operating System Programming Tools and Interfaces*.
2. IBM recommends that you process the object files of your application with **ld -r** to link them into one final object file that can still be linked with libraries. You can do this before creating the **installp** diskette.

The following example combines all of the `my*.o` object files into a final object file named `mypgm.o`, which can be shipped on an **installp** diskette:

```
ld -r -o mypgm.o my*.o
```

3. IBM recommends that you store the object files for your application in a directory named `/usr/lpp/pgm-name/bin`. This prevents the **installp** command from deleting them after installation.
4. Your **lpp.linkgsl** script should use **libgsl.a** if **libogsl.a** is not available.

Example **lpp.linkgsl** Shell Script

A typical **lpp.linkgsl** script might look like this:

```
# The current directory has already been set to /usr/lpp/mypgm

if [ -r /usr/lib/libogsl.a ]
then
    GSLLIB=ogsl
else
    # May be running on an older version of AIX that doesn't
    # have the shared library. Use the unshared version.
    GSLLIB=gsl
fi

# bin/mypgm.o was previously linked with:
# ld -r -o mypgm.o my*.o

cc -o mypgm bin/mypgm.o -l$GSLLIB
rc = $?
if [ $rc -ne 0 ]
then
    echo "Failed trying to relink mypgm." >&2
else
    # Move executable to /usr/bin, where users can run it
    cp mypgm /usr/bin/mypgm
    rm -f mypgm
fi
```

Using GSL in the X-Windows Environment

The GSL library, X-Windows Graphics Support Library (XGSL), allows the use of GSL routines in the X-Windows environment. XGSL is installed in the `/usr/lib` directory when you install GSL. To use GSL in X-Windows, applications must bind the XGSL library in place of `libgsl.a`. It is important to note that GSL and XGSL are mutually exclusive and cannot be used simultaneously.

The XGSL library is the `libxgsl.a`. A GSL application which is bound to XGSL runs as a client in the X-Windows environment. See the *IBM AIX X-Windows Programmer's Reference* for more information on bind procedures.

XGSL and GSL Differences

XGSL functions the same as GSL except for the following:

- Since the X-Windows system uses the cursor position to indicate which window is active, XGSL ignores calls to move the cursor (`gsmcur`). Applications can define the shape of the cursor while in the XGSL window, but X-Windows always controls its placement.
- Locator input is always returned in the tablet absolute position format regardless of the type of pointing device attached. In addition, the query locator routines `gsqloc` and `gsqlex` return the attributes for a tablet device, even if a mouse is attached to the system.
- Use of the `gslcat` function is restricted to setting tablet dead zones.
- XGSL ignores the `gslock`, `gsunlk`, `gssend` and `gspick` functions.
- XGSL ignores file descriptors passed by the `gsinit` call. Instead, XGSL opens a new window at initialization time.
- XGSL applications must use a ring buffer for input. XGSL does not support standard input. Attempts to condition standard in, out or error with `ioctl`s can produce unpredictable results. See "gsinit" on page 6-91 for more information.

Using XGSL

You can execute applications bound to the `libxgsl.a` from any Xterm currently in operation in the X-Windows environment. If no `.Xdefaults` file exists to define the window size, XGSL opens a full screen window for use by the application. Since the user defines the `.Xdefaults` file, applications should use `gsqdsp` to obtain the actual window size in *pels* (picture elements). It is then an application's responsibility to ensure that all coordinates are within the boundaries of the window.

Note: The coordinate system defines 0,0 as the lower left corner of a newly opened window.

When using the XGSL, library applications receive the standard **SIGGRANT** and **SIGRETRACT** signals. However, because applications are in the X-Windows environment, they receive these signals whenever an XGSL window is fully or partially obscured by another window. An application must stop drawing to the window when it receives a **SIGRETRACT**. It can resume when it receives the **SIGGRANT** signal.

Linking to the XGSL library

To link a GSL program to XGSL, create a link to **libxgsl.a**. instead of the standard **libgsl.a**. For example, to link the GSL program called **mygsl.c** to XGSL issue the following command **cc mygsl.c -lxgsl -lX11 -lpsd -lsock**.

The **a.out** produced is suitable for execution in X-Windows.

Creating a .Xdefaults file for use by XGSL

If a **.Xdefaults** file exists in the **\$HOME** directory, and it has an entry for the bound application, then XGSL opens a window of the specified height, width, x/y origin and border width. An entry for an application in the **.Xdefaults** file must have the following format:

```
<<appname>.Geometry: =<<width>x<<height>+-<<x>+-<<y>  
<<appname>.bwidth: =<<borderwidth>
```

Where:

```
<<appname> = the executable application that is bound to XGSL  
<<height> = the desired height of the window  
<<width> = the desired width of the window  
<<x> and <<y> = the desired placement of the lower left corner of the window  
<<borderwidth> = the desired width of the border
```

Please see *IBM AIX X-Windows Programmer's Reference* for complete details on using **.Xdefault** files.

gsbply

gsbply

Purpose

Defines the beginning of an area to fill.

C Syntax

int gsbply- ()

FORTRAN Syntax

INTEGER function gsbply

Pascal Syntax

FUNCTION gsbply- : INTEGER [PUBLIC];

Description

The **gsbply** subroutine defines the beginning of a two-dimensional shape or set of shapes to be filled.

The following output routines are valid between a **gsbply** call and a **gseply** call:

- Draw polyline (gspoly)
- Draw circle (gscir)
- Draw ellipse (gsell)
- Draw circular arc (gscarc or gscrca)
- Draw elliptical arc (gseara or gsearc)

Note: Any other subroutines used before the **gseply** subroutine is called do not become part of the shape or set of shapes to be filled, and can produce unpredictable results.

Before the fill occurs, the shapes drawn by each routine called between **gsbply** and **gseply** are connected. The first point of each shape is linked to the last point of the previous shape, and the last point of the last shape is linked to the first point of the first shape. The shapes may overlap to any degree but must share at least one common point between adjacent shapes.

Processing of the **SIGRETRACT** signal is postponed until the **gseply** subroutine, end of area to fill, is called.

See “gseply” on page 6-61 and “gspcls” on page 6-125 for related information.

The relevant attributes are:

- Color map
- Plane mask
- Fill color index
- Fill style
- Logical operation.

Return Value

GS_SUCC	Successful.
GS_USUC	Unsuccessful.

Related Information

In this book: “gseply” on page 6-61 and “gspcls” on page 6-125.

gscarc

gscarc

Purpose

Draws a circular arc between two points.

C Syntax

```
int gscarc_ (cx, cy, cr, bx, by, ex, ey)
```

```
int *cx, *cy, *cr, *bx, *by, *ex, *ey;
```

FORTRAN Syntax

```
INTEGER function gscarc (cx, cy, cr, bx, by, ex, ey)
```

```
INTEGER cx, cy, cr, bx, by, ex, ey
```

Pascal Syntax

```
FUNCTION gscarc_ (
```

```
  VAR cx, cy, cr, bx, by, ex, ey : INTEGER  
): INTEGER [PUBLIC];
```

Description

The **gscarc** subroutine draws a counterclockwise circular arc of the specified radius from the beginning point to the ending point. The radius is expressed in number of pixels.

The relevant attributes are:

- Color map
- Plane mask
- Line color index
- Line style
- Logical operation.

Parameters

<i>cx, cy</i>	Define the coordinates of the center of the circle. For displays, the center is restricted to -2048 to 2048. For printers and plotters, the center is restricted to screen coordinates.
<i>cr</i>	Defines the radius of the circle.
<i>bx, by</i>	Define the coordinates of the beginning point on the circle.
<i>ex, ey</i>	Define the coordinates of the ending point on the circle.

If the beginning and ending points are identical, a full circle is drawn.

Note that the application must control the accuracy of the end points (*bx, by* and *ex, ey*) when drawing circular arcs. If the start point of the arc and end point of the arc lie within one pixel of the true circle, the arc will be drawn successfully. Other values can cause the subroutine to fail. If the **gscarc** subroutine fails because of an inaccurate starting point, **GS-ASTR** is returned, while for an inaccurate ending point, **GS-AEND** is returned.

Return Value

GS-SUCC	Successful.
GS-CORD	Invalid coordinate.
GS-RDUS	Invalid radius specification.
GS-INAC	Virtual terminal inactive.
GS-AEND	Invalid end point.
GS-ASTR	Invalid start point.

gscatt

gscatt

Purpose

Sets the attributes of the single-color cursor.

C Syntax

int gscatt_ (*color, width, height, pattern, 0x, 0y*)

int **color, *width, *height, *pattern, *0x, *0y;*

FORTRAN Syntax

INTEGER function gscatt (*color, width, height, pattern, 0x, 0y*)

INTEGER *color, width, height, pattern, 0x, 0y*

Pascal Syntax

FUNCTION gscatt_ (

VAR *color, width, height*: **INTEGER**;

pattern: **ARRAY** [1..*k*] of **INTEGER**;

0x, 0y: **INTEGER**

): **INTEGER** [**PUBLIC**];

Description

The **gscatt** subroutine defines the single-color cursor for the GSL. The **gsmcap** subroutine must initialize the color map before **gscatt** can be called.

Only one cursor, either the single-color cursor or the multicolor cursor, can be active in the GSL at any one time. The **gscatt** subroutine forces all subsequent calls to the **gsmcur** and **gsecur** subroutines to operate on the single-color version of the cursor. To change from the multicolor cursor to the single-color cursor, erase the cursor with **gsecur**, then call the **gscatt** subroutine.

Parameters

<i>color</i>	Refers to an entry in the color map. If the index value is -1, the attribute is unchanged.
<i>width, height</i>	Define, in pixels, the width and height of the bit pattern to be used as the cursor. If <i>width</i> or <i>height</i> equals -1, then the pattern remains unchanged.
<i>pattern</i>	Defines the image used as a cursor. The ceiling ($\text{width}/32$) indicates the number of words per row and <i>height</i> indicates the number of rows. The cursor data must be supplied in row (scan line) major order. If <i>width</i> implies partial use of a word, the rest of the word is unused. To fully define the cursor pattern, <i>pattern</i> should be $(\text{ceiling}(\text{width}/32) \times \text{height})$ words in length.
<i>0x, 0y</i>	Indicate the origin of the cursor relative to the lower leftmost corner (0, 0) of the cursor pattern. The origin must be placed within the cursor pattern: $0x < \text{width}$ and $0y < \text{height}$. The origin of the cursor is placed at the position indicated, when the application moves the cursor using the gsmcur subroutine. If <i>x</i> equals -1, then the origin remains unchanged.

The maximum size of the cursor is device-dependent and can be determined by using the **gsqdsp** subroutine.

You cannot change the cursor attributes while the cursor is visible.

There is no default cursor defined, so all cursor parameters must be set before the cursor is displayed.

For Pascal, the application must declare the array passed as being fixed length and declare the routine as accepting arrays of that length. The *k* in the routine declaration must be a constant.

Return Value

GS-SUCC	Successful.
GS-COLI	Invalid color index.
GS-CURS	Cursor size invalid.
GS-CURO	Cursor origin invalid.
GS-CURV	Cursor visible.

Related Information

In this book: “gsecur” on page 6-56, “gsmcat” on page 6-113, and “gsmcur” on page 6-116.

gscenv

Purpose

Converts a circular arc or full circle into a polyline.

C Syntax

```
int gscenv- (cx, cy, cr, bx, by, ex, ey, len, x, y, pre)
int *cx, *cy, *cr, *bx, *by, *ex, *ey, *len, *x, *y, *pre;
```

FORTRAN Syntax

```
INTEGER function gscenv (cx, cy, cr, bx, by, ex, ey, len, x, y, pre)
INTEGER cx, cy, cr, bx, by, ex, ey, len, x(*), y(*), pre
```

Pascal Syntax

```
FUNCTION gscenv- (
  VAR cx, cy, cr, bx, by, ex, ey, len: INTEGER;
  VAR x, y: ARRAY [1..k] of INTEGER;
  VAR pre: INTEGER
): INTEGER [PUBLIC];
```

Description

The **gscenv** subroutine converts a counterclockwise circular arc definition into an array of vertices. The list of vertices can then be used to draw a circular arc with the **gspoly** subroutine or to fill a circular arc with the **gsfply** subroutine. In general, it can be concatenated with other list(s) of vertices to draw or fill more complex shapes, such as chord arcs, pie arcs, and rectangles with rounded corners.

When beginning and ending points are identical, the list of vertices contains the full circle, which can then be drawn or filled.

Parameters

<i>cx, cy</i>	Define the coordinates of the center of the circle.
<i>cr</i>	Defines the radius of the circle, which must not be equal to zero. If <i>cr</i> is negative, it is automatically converted to a positive value for use by the subroutine.
<i>bx, by</i>	Define the coordinates of the beginning point of the arc.
<i>ex, ey</i>	Define the coordinates of the ending point of the arc.
<i>len</i>	Defines the number of points in the coordinate <i>x</i> and <i>y</i> arrays. It must be numerically at least one greater than the value contained in the precision parameter, but not less than 65.
<i>x, y</i>	Define, as coordinate arrays, the vertices that represent the circular shape when drawn or filled.
<i>pre</i>	Defines precision level, which specifies the maximum number of line segments that can be generated for a full circle. The number of line segments actually generated depends on the size of the circle.

There are four levels of precision that can be requested:

- 64 (65 vertices)
- 128 (129 vertices)
- 256 (257 vertices)
- 512 (513 vertices).

Therefore, $len \geq pre + 1$.

All other precision values are reserved and must not be used, as their results are unpredictable. The default value for *pre* is 64.

The subroutine allows ample leniency toward the accuracy of the specification of the beginning and ending points. The arc of the specified radius will always start and end exactly at the specified points.

If the beginning and ending points are identical, a full circle of the specified radius is generated.

When the subroutine is invoked, the length parameter must contain the maximum number of entries in the *x* and *y* arrays. If erroneous conditions arise, *len* is set to zero. Under normal conditions, *len* specifies the number of vertices returned by the subroutine in the *x* and *y* arrays.

For Pascal, the application must declare the arrays passed as being fixed length and declare the routine as accepting arrays of that length; the *k* in the routine declaration must be a constant.

Return Value

GS_SUCC	Successful.
GS_CORD	Invalid coordinate.
GS_NCOR	Invalid number of coordinates.

gscir

gscir

Purpose

Draws a circle.

C Syntax

```
int gscir_ (cx, cy, cr)
```

```
int *cx, *cy, *cr;
```

FORTRAN Syntax

```
INTEGER function gscir (cx, cy, cr)
```

```
INTEGER cx, cy, cr
```

Pascal Syntax

```
FUNCTION gscir_ (
```

```
  VAR cx, cy, cr: INTEGER  
): INTEGER [PUBLIC];
```

Description

The **gscir** subroutine draws a circle of the specified radius. The radius is expressed in number of pixels.

The relevant attributes are:

- Color map
- Plane mask
- Line color index
- Line style
- Logical operation.

Parameters

cx, cy Define the coordinates of the center of the circle.
cr Defines the radius of the circle.
If the radius is zero, a single point is drawn at the center.

Return Value

GS_SUCC Successful.
GS_CORD Invalid coordinate.
GS_RDUS Invalid radius specification.
GS_INAC Virtual terminal inactive.

gsclrs

gsclrs

Purpose

Clears the screen, filling it with the background color.

C Syntax

```
int gsclrs_ ( )
```

FORTRAN Syntax

INTEGER function **gsclrs**

Pascal Syntax

FUNCTION **gsclrs_**: INTEGER [PUBLIC];

Description

The **gsclrs** subroutine fills the frame buffer with the background color (color index zero).

The relevant attribute is:

- Color map.

For printers, the **gsclrs** subroutine forces pending graphics to be printed, advances the paper to a new page, and purges the print buffer.

For plotters, the **gsclrs** subroutine forces pending graphics to be *displayed*, and issues a prompt to the active screen (console) requesting that the paper be changed.

Return Value

GS_SUCC	Successful.
GS_INAC	Virtual terminal inactive.

gscmap

Purpose

Specifies the color mapping.

C Syntax

```
int gscmap- (number, red, green, blue)
```

```
int *number, *red, *green, *blue;
```

FORTRAN Syntax

```
INTEGER function gscmap (number, red, green, blue)
```

```
INTEGER number, red (*), green (*), blue (*),
```

Pascal Syntax

```
FUNCTION gscmap- (
```

```
VAR number INTEGER;
```

```
VAR red, green, blue: ARRAY [0..k] of INTEGER
```

```
): INTEGER [PUBLIC];
```

Description

The **gscmap** subroutine specifies the mapping between the color index attribute and the color it produces on the display.

The default color table mapping for the first 16 colors is the same as the default color map attributes in KSR mode. The remaining color values are initialized in a hardware-dependent manner.

Parameters

number Indicates how many colors the input intensity arrays contain.

red, green, blue Define arrays that contain the intensity levels of the corresponding color. Each entry in an array specifies the intensity value for the corresponding color index.

The value in each entry for the *red*, *green*, and *blue* intensity arrays is between 0x0000, representing zero intensity, and 0x3FFF, representing full intensity. The following additional increments of intensity are possible, depending on the adapter hardware in use:

0x2000	1/2 intensity
0x1000	1/4 intensity
0x0800	1/8 intensity
0x0400	1/16 intensity
0x0200	1/32 intensity
0x0100	1/64 intensity.

Combinations of these values can be used to create intermediate levels of intensity. For example, 0xC000 gives 3/16 intensity, while 0x3000 gives 3/4 intensity.

The actual number of bits from bit 13 to bit 0 that affect the color on the display is dependent on the number of bits in the digital-to-analog converter of the adapter hardware in use. This size information is available by using the **gsqdsp** subroutine.

An application cannot change a single arbitrary color entry in the color map (or the VLT). It must change all the entries for all the colors up to and including the desired entry.

For Pascal, the application must declare the arrays passed as being fixed length and declare the routine as accepting arrays of that length; the *k* in the routine declaration must be a constant.

Return Value

GS_SUCC	Successful.
GS_TABL	Invalid table length.
GS_INAC	Virtual terminal inactive.

gscrca

Purpose

Draws a circular arc between two angles.

C Syntax

```
int gscrca_ (cx, cy, cr, ba, ea)
```

```
int *cx, *cy, *cr, *ba, *ea;
```

FORTRAN Syntax

```
INTEGER function gscrca (cx, cy, cr, ba, ea)
```

```
INTEGER cx, cy, cr, ba, ea
```

Pascal Syntax

```
FUNCTION gscrca_ (
```

```
  VAR cx, cy, cr, ba, ea : INTEGER  
): INTEGER [PUBLIC];
```

Description

The **gscrca** subroutine draws a counterclockwise circular arc of the specified radius from the beginning point as defined by an angle specification to the ending point as defined by an angle specification.

The angle specifications are given in tenths of degrees, from 0 to 3600. Values outside this range cause the **gscrca** subroutine to fail.

The relevant attributes are:

- Color map
- Plane mask
- Line color index
- Line style
- Logical operation.

Parameters

<i>cx, cy</i>	Define the coordinates of the center of the circle. For displays, the center is restricted to -2048 to 2048. For printers and plotters, the center is restricted to screen coordinates.
<i>cr</i>	Defines the radius of the circle in device coordinates.
<i>ba</i>	Defines the start point of the circular arc as an angle in tenths of degrees, from 0 to 3600.
<i>ea</i>	Defines the end point of the circular arc as an angle in tenths of degrees, from 0 to 3600.

If the beginning and ending angles are identical, a full circle is drawn.

Return Value

GS-SUCC	Successful.
GS-ANGL	Invalid angle.
GS-RDUS	Invalid radius specification.
GS-CORD	Invalid coordinate.
GS-INAC	Virtual terminal inactive.

gsdjply

Purpose

Draws a polyline, a set of lines that connects a sequence of points.

C Syntax

```
int gsdjply- (polylines, points, x, y)
```

```
int *polylines, *points, *x, *y;
```

FORTRAN Syntax

```
INTEGER function gsdjply (polylines, points, x, y)
```

```
INTEGER polylines, points, x (*), y (*)
```

Pascal Syntax

```
FUNCTION gsdjply- (
```

```
  VAR polylines: INTEGER;  
  VAR points: ARRAY [1..k] of INTEGER;  
  VAR x, y: ARRAY [1..l] of INTEGER  
): INTEGER [PUBLIC];
```

Description

The **gsdjply** subroutine draws a series of polylines as a set of lines, as defined by the current relevant attributes.

The relevant attributes are:

- Color map
- Plane mask
- Line color index
- Line style
- Logical operation.

Parameters

<i>polylines</i>	Defines the number of polylines to draw. This value must be ≥ 1 .
<i>points</i>	Defines the number of points in each polyline. This value must be ≥ 2 .
<i>x, y</i>	Define, as an array, the points for line drawing.

For Pascal, the application must declare the arrays passed as being fixed length and declare the routine as accepting arrays of that length. The *k* and *l* in the routine declaration must be constants.

Return Value

GS_SUCC	Successful.
GS_CORD	Invalid coordinate.
GS_NCOR	Invalid number of coordinates.
GS_INAC	Virtual terminal inactive.

gsdpik

Purpose

Defines the closing delimiter for a group of GSL output functions.

C Syntax

`int gsd pik- ()`

FORTRAN Syntax

`INTEGER function gsd pik`

Pascal Syntax

`FUNCTION gsd pik- : INTEGER [PUBLIC];`

Description

The **gsdpik** subroutine defines the closing delimiter for a group of pickable output functions. The output function calls that precede this command cause a pick input from the display adapter if any vertices intersect a pick aperture (window).

The **gsdpik** subroutine is provided only for use with the IBM 5081 Display, and not for use with other displays.

See “gsepik” on page 6-59 and the list of GSL output subroutines on page 6-16 for related information.

The relevant attributes are:

- Color map
- Plane mask
- Fill color index
- Fill style
- Logical operation.

gsdpik

Return Value

GS_SUCC	Successful.
GS_USUC	Unsuccessful.

Related Information

In this book: “gsepik” on page 6-59 and the list of GSL output subroutines on page 6-16.

gseara

Purpose

Draws an elliptical arc between two angles.

C Syntax

```
int gseara_ (cx, cy, ma, mi, ang, sa, ea)
```

```
int *cx, *cy, *ma, *mi, *ang, *sa, *ea;
```

FORTRAN Syntax

```
INTEGER function gseara (cx, cy, ma, mi, ang, sa, ea)
```

```
INTEGER cx, cy, ma, mi, ang, sa, ea
```

Pascal Syntax

```
FUNCTION gseara_ (
```

```
VAR cx, cy, ma, mi, ang, sa, ea : INTEGER  
): INTEGER [PUBLIC];
```

Description

The **gseara** subroutine draws a counterclockwise elliptical arc of the specified axes and angle from the beginning point defined by an angle specification to the ending point defined by an angle specification. The axes are expressed in number of pixels.

The angle specifications are given in tenths of degrees, from 0 to 3600. Values outside this range cause the **gseara** subroutine to fail.

The relevant attributes are:

- Color map
- Plane mask
- Line color index
- Line style
- Logical operation.

Parameters

<i>cx, cy</i>	Define the coordinates of the center of the ellipse. For displays, the center is restricted to -2048 to 2048. For printers and plotters, the center is restricted to screen coordinates.
<i>ma, mi</i>	Define half of the nonzero major and minor axes of the ellipse.
<i>ang</i>	Defines the angle between the major axis and the x-axis. If <i>ang</i> is zero, the major axis is on the x-axis and the minor axis is on the y-axis. The angle is expressed in tenths of degrees, from 0 to 3600.
<i>sa</i>	Defines the angle of the starting point of the elliptical arc, measured counterclockwise from the major axis. The angle is expressed in tenths of degrees, from 0 to 3600.
<i>ea</i>	Defines the angle of the ending point of the elliptical arc, measured counterclockwise from the major axis. The angle is expressed in tenths of degrees, from 0 to 3600.

If the beginning and ending points are identical, a full ellipse is drawn.

Return Value

GS-SUCC	Successful.
GS-CORD	Invalid coordinate.
GS-ELMM	Invalid major or minor axis.
GS-INAC	Virtual terminal inactive.
GS-ANGL	Invalid angle.
GS-NMEM	Insufficient resources.

gsecnv

Purpose

Converts an ellipse to a polyline.

C Syntax

```
int gsecnv- (cx, cy, ma, mi, ang, bx, by, ex, ey, rot, len, x, y, pre)
int *cx, *cy, *ma, *mi, *ang, *bx, *by, *ex, *ey, *rot, *len, *x, *y, *pre;
```

FORTRAN Syntax

```
INTEGER function gsecnv (cx, cy, ma, mi, ang, bx, by, ex, ey, rot, len, x, y, pre)
INTEGER cx, cy, ma, mi, ang, bx, by, ex, ey, rot, len, x(*), y(*), pre
```

Pascal Syntax

```
FUNCTION gsecnv- (
  VAR cx, cy, ma, mi, ang, bx, by, ex, ey, rot, len: INTEGER;
  VAR x, y: ARRAY [1..k] of INTEGER;
  VAR pre: INTEGER
): INTEGER [PUBLIC];
```

Description

The **gsecnv** subroutine converts a counterclockwise elliptical arc definition into an array of vertices. The list of vertices can then be used to draw an elliptical arc with the **gspoly** subroutine or to fill an elliptical arc with the **gsfply** subroutine. In general, it can be concatenated with other list(s) of vertices to draw or fill more complex shapes, such as chord arcs, pie arcs, or rectangles with round corners.

When the beginning and ending points are identical, the list of vertices contains the full ellipse, which can then be drawn or filled.

Parameters

<i>cx, cy</i>	Define the coordinates of the center of the ellipse.
<i>ma, mi</i>	Define half of the nonzero major and minor axes of the ellipse.
<i>ang</i>	Defines the off-axis angle of the ellipse. If <i>ang</i> is zero, the major axis is the x-axis and the minor axis is the y-axis. A positive value rotates the ellipse counterclockwise; a negative value rotates it clockwise. All values are in degrees and modulo 360.
<i>bx, by</i>	Define the coordinates of the beginning point of the arc.
<i>ex, ey</i>	Define the coordinates of the ending point of the arc.
<i>rot</i>	<p>Specifies whether the application must perform rotational transformation. Possible settings are:</p> <ol style="list-style-type: none">0 The coordinates of the beginning and ending points passed by the application correspond to an arc of an orthogonal ellipse. No rotational transformation is performed, thus improving performance.1 The beginning and ending points are transformed by the application and lie on the off-axis ellipse. <p>All other values are reserved and must not be used, as they can produce unpredictable results.</p>
<i>len</i>	Defines the number of points in the coordinate <i>x</i> and <i>y</i> arrays. It must be numerically at least one greater than the value contained in the precision parameter and greater than or equal to 65.
<i>x, y</i>	Define, as coordinate arrays, the vertices that represent the elliptical shape when drawn or filled.
<i>pre</i>	<p>Defines precision level, which specifies the maximum number of line segments that can be generated for a full ellipse. The number of line segments actually generated depends on the size of the ellipse.</p> <p>There are four levels of precision that can be requested:</p> <ul style="list-style-type: none">• 64 (65 vertices)• 128 (129 vertices)• 256 (257 vertices)• 512 (513 vertices).

Therefore, $len \geq pre + 1$.

All other precision values are reserved and must not be used, as their results are unpredictable. The default value for *pre* is 64.

The subroutine allows ample leniency toward the accuracy of the specification of the beginning and ending points. The arc of the specified angle always starts and ends exactly at the specified points. If the beginning and ending points are identical, a full ellipse of the specified angle is generated.

When the subroutine is invoked, the length parameter must contain the maximum number of entries in the x and y arrays. If erroneous conditions arise, *len* is set to zero. Under normal conditions, *len* specifies the number of vertices returned by the subroutine in the x and y arrays.

For Pascal, the application must declare the arrays passed as being fixed length and declare the routine as accepting arrays of that length; the k in the routine declaration must be a constant.

Return Value

GS_SUCC	Successful.
GS_CORD	Invalid coordinate.
GS_NCOR	Invalid number of coordinates.

gsecur

gsecur

Purpose

Erases all enabled cursors and makes them invisible.

C Syntax

```
int gsecur_ ( )
```

FORTRAN Syntax

```
INTEGER function gsecur
```

Pascal Syntax

```
FUNCTION gsecur_: INTEGER [PUBLIC];
```

Description

The **gsecur** subroutine makes the enabled cursors invisible.

For adapters with hardware cursor support, **gsecur** simply turns off the cursor or cursors. Otherwise, **gsecur** reverses the actions taken to place the cursor or cursors in the frame buffer.

Return Value

GS_SUCC	Successful.
GS_INAC	Virtual terminal inactive.

gsearc

Purpose

Draws an elliptical arc between two points.

C Syntax

```
int gsearc_ (cx, cy, ma, mi, ang, bx, by, ex, ey, rot)
int *cx, *cy, *ma, *mi, *ang, *bx, *by, *ex, *ey, *rot;
```

FORTRAN Syntax

```
INTEGER function gsearc (cx, cy, ma, mi, ang, bx, by, ex, ey, rot)
INTEGER cx, cy, ma, mi, ang, bx, by, ex, ey, rot
```

Pascal Syntax

```
FUNCTION gsearc_ (
  VAR cx, cy, ma, mi, ang, bx, by, ex, ey, rot : INTEGER
): INTEGER [PUBLIC];
```

Description

The **gsearc** subroutine draws a counterclockwise elliptical arc of the specified axes and angle from the beginning point to the ending point. The axes are expressed in number of pixels.

The angle specifications are given in tenths of degrees, from 0 to 3600. Values outside this range cause the **gsearc** subroutine to fail.

The relevant attributes are:

- Color map
- Plane mask
- Line color index
- Line style
- Logical operation.

Parameters

<i>cx, cy</i>	Define the coordinates of the center of the ellipse. For displays, the center is restricted to -2048 to 2048. For printers and plotters, the center is restricted to screen coordinates.
<i>ma, mi</i>	Define half of the nonzero major and minor axes of the ellipse.
<i>ang</i>	Defines the angle between the major axis and the x-axis. If <i>ang</i> is zero, the major axis is on the x-axis and the minor axis is on the y-axis. The angle is expressed in tenths of degrees, from 0 to 3600.
<i>bx, by</i>	Define the coordinates of the beginning point on the ellipse.
<i>ex, ey</i>	Define the coordinates of the ending point on the ellipse.
<i>rot</i>	Specifies whether the application must perform rotational transformation. Possible settings are: 0 The coordinates of the beginning and ending points passed by the application correspond to an arc of an orthogonal ellipse. No rotational transformation is performed, thus improving performance. 1 The beginning and ending points are transformed by the application and lie on the off-axis ellipse. All other values are reserved and must not be used, as they can produce unpredictable results.

If the beginning and ending points are identical, regardless of whether or not they are on the ellipse, a full ellipse is drawn.

Return Value

GS_SUCC	Successful.
GS_CORD	Invalid coordinate.
GS_ELM	Invalid major or minor axis.
GS_INAC	Virtual terminal inactive.
GS_ANG	Invalid angle.
GS_NMEM	Insufficient resources.
GS_AEND	Invalid end point.
GS_ASTR	Invalid start point.

gsell

Purpose

Draws an ellipse.

C Syntax

```
int gsell_ (cx, cy, ma, mi, ang)
```

```
int *cx, *cy, *ma, *mi, *ang;
```

FORTRAN Syntax

```
INTEGER function gsell (cx, cy, ma, mi, ang)
```

```
INTEGER cx, cy, ma, mi, ang
```

Pascal Syntax

```
FUNCTION gsell_ (
```

```
  VAR cx, cy, ma, mi, ang : INTEGER  
): INTEGER [PUBLIC];
```

Description

The **gsell** subroutine draws an ellipse of the specified axes and angle. The axes are expressed in number of pixels.

The angle specifications are given in tenths of degrees, from 0 to 3600. Values outside this range cause the **gsell** subroutine to fail.

The relevant attributes are:

- Color map
- Plane mask
- Line color index
- Line style
- Logical operation.

gsell

Parameters

<i>cx, cy</i>	Define the coordinates of the center of the ellipse.
<i>ma, mi</i>	Define half of the nonzero major and minor axes of the ellipse.
<i>ang</i>	Defines the angle between the major axis and the x-axis. If it is zero, the major axis is on the x-axis and the minor axis is on the y-axis. The angle is expressed in tenths of degrees, from 0 to 3600.

Return Value

GS_SUCC	Successful.
GS_CORD	Invalid coordinate.
GS_ELMM	Invalid major or minor axis.
GS_INAC	Virtual terminal inactive.
GS_ANGL	Invalid angle.
GS_NMEM	Insufficient resources.

gsepik

Purpose

Defines the beginning delimiter for a group of GSL output primitive functions.

C Syntax

```
int gsepik- (pickwind)  
int *pickwind;
```

FORTRAN Syntax

```
INTEGER function gsepik (pickwind)  
INTEGER pickwind
```

Pascal Syntax

```
FUNCTION gsepik- (  
  VAR pickwindow : INTEGER  
): INTEGER [PUBLIC];
```

Description

The **gsepik** subroutine defines the opening delimiter for a group of output routines. The GSL output functions that occur after this command cause a pick input from the display adapter for each intersection of a vertex and the pick aperture window centered about the current cursor position.

The pick input is placed on the user input ring, indicating the count of output functions since the start of the pick. The input also indicates the center of the pick window in x,y coordinates.

The **gsepik** subroutine is provided only for use with the IBM 5081 Display, and not for use with other displays.

See "gsdpik" on page 6-47 and the list of GSL output subroutines on page 6-16 for related information.

gsepik

Parameters

pickwind Defines a pick window center about the current cursor position. Valid values are between 1 and the maximum cursor size.

A cursor must be defined and visible for the **gsepik** function to operate.

Processing of the **SIGRETRACT** signal is suspended until the completion of the pick function with the **gsdpik** subroutine.

Return Value

GS_SUCC	Successful.
GS_NCOR	Undefined cursor.
GS_CURV	Cursor not visible.
GS_IVWD	Invalid pick window size.
GS_USUC	Unsuccessful.

Related Information

In this book: “gsdpik” on page 6-47 and the list of GSL output subroutines on page 6-16.

gseply

Purpose

Defines the end of an area to fill.

C Syntax

`int gseply- ()`

FORTRAN Syntax

`INTEGER function gseply`

Pascal Syntax

`FUNCTION gseply- : INTEGER [PUBLIC];`

Description

The **gseply** subroutine defines the end of a two-dimensional shape or set of shapes to be filled, then fills each of the valid primitives drawn since the last **gspcls** or **gsbply** subroutine was called.

See “gsbply” on page 6-28 and “gspcls” on page 6-125 for related information.

The relevant attributes are:

- Color map
- Plane mask
- Fill color index
- Fill style
- Logical operation.

gseply

Return Value

GS-SUCC	Successful.
GS-USUC	Unsuccessful.

Related Information

In this book: “gsbply” on page 6-28 and “gspcls” on page 6-125.

gsevds

Purpose

Disables the reporting of events.

C Syntax

```
int gsevds_ (event)
```

```
int *event;
```

FORTRAN Syntax

```
INTEGER function gsevds (event)
```

```
INTEGER event
```

Pascal Syntax

```
FUNCTION gsevds_ (
```

```
  VAR event: INTEGER  
): INTEGER [PUBLIC];
```

Description

The **gsevds** subroutine disables the reporting of events of a given type. When the keyboard event is disabled, the keyboard is locked and no keystroke input is placed in the input ring buffer. Similarly, for all other devices, if an event is disabled, the device producing the event is inhibited from placing input into the ring.

A valid input ring must be defined during the GSL initialization.

Parameters

event

The recognized events are as follows:

- 1 Keystroke
- 3 Locator movement or button
- 4 Lighted Program Function Key (LPFK)
- 5 Valuator

The user can enable the keyboard by keying the sequence **Esc B** (the ANSI Enable Manual Input). After this sequence, keystroke events are again reported.

Return Value

GS_SUCC	Successful.
GS_EVNT	Invalid event type.
GS_UNSC	Unsuccessful.

gseven

Purpose

Enables the reporting of events.

C Syntax

```
int gseven- (event)
```

```
int *event;
```

FORTRAN Syntax

```
INTEGER function gseven (event)
```

```
INTEGER event
```

Pascal Syntax

```
FUNCTION gseven- (
```

```
  VAR event: INTEGER  
): INTEGER [PUBLIC];
```

Description

The **gseven** subroutine enables the reporting of events of a given type. If the device producing the event is enabled, then **gseven** lets it put data into the ring buffer. If the event type is not recognized, no action is taken.

A valid input ring must be defined during the GSL initialization.

Parameters

event

The recognized events are as follows:

- 1 Keystroke
- 3 Locator movement or button
- 4 Lighted Program Function Key (LPFK)
- 5 Valuator

gseven

After GSL initialization, only the keyboard is enabled. If the application wishes the other input devices enabled, it must explicitly enable them with this command.

Return Value

GS_SUCC	Successful.
GS_EVNT	Invalid event type.
GS_UNSC	Unsuccessful.

gsevt

Purpose

Waits for an input event.

C Syntax

```
int gsevt_ (wait, data)
```

```
int *wait, data[13];
```

FORTRAN Syntax

```
INTEGER function gsevt (wait, data)
```

```
INTEGER wait, data (13)
```

Pascal Syntax

```
FUNCTION gsevt_ (
```

```
VAR wait: INTEGER;
```

```
VAR data: ARRAY [0..12] of INTEGER
```

```
): INTEGER [PUBLIC];
```

Description

The **gsevt** subroutine returns the relevant information for the oldest input event in the ring buffer.

The function works as follows:

- If an event is in the ring, then **gsevt** parses the oldest event in the ring. It returns the event type and its data in the buffer provided by the application.
- If no event is in the ring and the application requested no wait, **gsevt** returns immediately. If the application requested a wait, the process execution is suspended until an enabled input event occurs; then **gsevt** returns the event type and its data in the buffer provided.

Warning: **gsevw** uses the application buffer passed to it for temporary storage. If the user has explicitly keyed part of an ANSI control sequence when the application calls **gsevw** with no wait request, then **gsevw** finds a partial event in the ring and leaves part of the parsed data for the event in the application buffer; however, **gsevw** returns a time-out event class. Unless the application returns the same unmodified buffer, or a different buffer containing identical information, the results of the next call to **gsevw** will be incorrect.

A valid input ring must be defined during the GSL initialization.

Parameters

wait

Determines whether or not to wait for an event. If *wait* is 0, then **gsevw** does not wait for an event if no event is available.

data

Specifies the location where GSL is to store the input data (up to 13 words). The *data* must be word aligned:

The possible events are:

- **Keystroke(s)**

This event type occurs when the user types a single graphic character or a single-byte control character. For these two events, **gsevw** returns a null-terminated byte string representing the graphic or control character that was typed. This event may also occur if the user has explicitly keyed an ANSI escape sequence; **gsevw** returns 2 bytes, the **Esc** and the next character in the sequence.

The *data* consists of a null-terminated ASCII string and is structured as follows:

Byte 0	Byte 1	Byte 2	Byte 3
Event type = 1			
Reserved			
code	code	code	code
:			
code	code	0	unused

- **Control sequence**

This event type indicates an ANSI control sequence, which is of the form:

ESC [*p* ; *p* ; . . . *p* *f*

where ESC is the ASCII escape character, *p* represents a parameter (one or more ASCII digits), the ellipsis represents additional parameters separated by semicolons, and *f* represents the final character that terminates the sequence (ASCII **a-z** or **A-Z**).

The ANSI control sequence occurs when the user presses a program function key on the keyboard or if the user enters an explicit control sequence.

The data consists of the parsed control sequence information. The Final Character is the valid or invalid final character. The Count indicates the number of parameters in the control sequence, with a maximum count of 10. These fields are followed by the Parameters. The *data* is structured as follows:

Byte 0	Byte 1	Byte 2	Byte 3
Event type = 2			
Final Character			
Count			
Parameter [1]			
:			
Parameter [Count]			

• Locator

This event indicates the user has moved the locator or pressed a button on the locator.

The data consists of locator position and status information. The X value and the Y value field contain a relative movement (delta x, delta y) for a mouse and an absolute position (x, y) for a tablet. The timestamp, which is elapsed time since system startup (IPL), is in sixtieths of a second.

The Buttons field contains the locator button status. For a mouse, each bit corresponds to a button, the most significant bit representing Button 1. A bit set to 1 indicates that the corresponding button is pressed. For a tablet, the most significant five bits represent the button pressed, according to the following scheme:

Status	Button
0	None pressed
1	Cursor upper left, stylus tip
2	Cursor upper right
3	Cursor lower left
4	Cursor lower right

For a tablet, the sixth most significant bit of the Buttons field indicates that the sensor is on (bit set) or off (bit not set).

The Type field contains a 0 if the locator is a mouse and a 1 if the locator is a tablet. The *data* is structured as follows:

Byte 0	Byte 1	Byte 2	Byte 3
Event type = 3			
X value			
Y value			
Type			
Buttons			
Timestamp			

- **LPFK**

This event type occurs when the user presses a key on the LPFK.

The data consists of the LPFK information. The LPFK field contains the decimal number of the LPFK pressed by the user, that is, 0 through 31. The Timestamp (time since system startup) is in sixtieths of a second. The *data* is structured as follows:

Byte 0	Byte 1	Byte 2	Byte 3
Event type = 4			
LPFK			
Timestamp			

- **Valuator**

This event type occurs when the user turns a valuator dial.

The data consists of the valuator information. The Valuator field contains the decimal number, 0 through 7, of the valuator turned by

the user. The Valuator Delta field contains the difference between the current valuator value and the last valuator value. The delta for a full turn is 256 for the IBM Valuator. The delta is positive for clockwise rotation and negative for counterclockwise rotation. The Timestamp (time since system startup) is in sixtieths of a second. The *data* is structured as follows:

Byte 0	Byte 1	Byte 2	Byte 3
Event type = 5			
Valuator			
Valuator Delta			
Timestamp			

- **Key code**

This event type occurs when the virtual terminal is in non-translated mode *and* a keyboard key is pressed, held down, or released. The *data* is structured as follows:

Byte 0	Byte 1	Byte 2	Byte 3
Event type = 6			
Key Position Code			
Key Scan Code			
Status			

Key position codes are found under "keyboard" on page 5-97. Status bits are found under "Input" on page 5-73.

- **Pick event**

This event type occurs while the pick operation is enabled and graphics primitives are being sent to the adapter. The *data* is structured as follows:

Byte 0	Byte 1	Byte 2	Byte 3
Event type = 7			
Pick Count			
Pick Center x			
Pick Center y			

A pick event code is generated when a structure traversal occurs. The pick occurs when pixels are determined to intersect the pick window (defined by the pick enable window size). The detection mode is always immediate, so that an event is generated as soon as an event occurs.

The pick event type is provided only for use with the IBM 5081 Display Adapter, and not for use with other displays.

- **Time out**

No data is returned.

It is important to note that **gsevwt** does not detect ANSI escape sequences. However, with the default virtual terminal keyboard mapping, it is not possible to generate an escape sequence by pressing a single key. Because **gsevwt** does parse ANSI control sequences, the routine cannot consider the press of the **Esc** key an event, so the routine waits for the next character to decide if the escape implies the start of a control sequence. Only if the next character is not the left bracket does **gsevwt** return the escape and the next character.

If the return code indicates overflow, the most recent input events from enabled devices are lost.

Return Value

GS_SUCC	Successful.
GS_ROVR	Ring buffer overflow.
GS_UDRG	Ring undefined.
GS_PARM	Too many control sequence parameters.
GS_ICTL	Invalid final character.

Related Information

In this book: "keyboard" on page 5-97 and "Input" on page 5-73.

gsfatt

gsfatt

Purpose

Sets the fill attributes.

C Syntax

```
int gsfatt_ (color, pattern, reserved)
```

```
int *color, *pattern, *reserved;
```

FORTRAN Syntax

```
INTEGER function gsfatt (color, pattern, reserved)
```

```
INTEGER color, pattern, reserved
```

Pascal Syntax

```
FUNCTION gsfatt_ (
```

```
  VAR color, pattern, reserved: INTEGER  
): INTEGER [PUBLIC];
```

Description

The **gsfatt** subroutine defines the attributes for the class of fill functions, which includes **gsfci**, **gsfell**, **gsfrec**, and **gsfp**.

Parameters

<i>color</i>	Refers to an entry in the color map. If <i>color</i> is -1, the attribute is unchanged. The default color after initialization is 15.
--------------	---

pattern Contains a value from the following list:

Value	Display	Printer or Plotter
-1	No change	No change
0	Solid	Solid
1	Horizontal lines	Narrow right diagonal lines
2	Vertical lines	Medium right diagonal lines
3	135-degree lines	Wide right diagonal lines
4	45-degree lines	Narrow diagonal cross-hatched
5	Cross-hatched (horizontal and vertical lines)	Medium diagonal cross-hatched
6	Cross-hatched (45- and 135-degree lines)	Wide diagonal cross-hatched

The default pattern is solid (0).

Some printers and plotters support additional fill patterns that can be selected with a *pattern* index greater than 6. If the device you are using does not support additional fill patterns and you specify a *pattern* index greater than 6, then the **gsfatt** subroutine returns the value **GS_SYLI**.

The fill pattern does not meet the border of the filled area on printers and plotters.

reserved

Represents a parameter that **gsfatt** ignores.

Return Value

GS_SUCC	Successful.
GS_COLI	Invalid color index.
GS_SYLI	Invalid style index.

gsfci

gsfci

Purpose

Fills a circle.

C Syntax

```
int gsfci_ (cx, cy, cr)
```

```
int *cx, *cy, *cr;
```

FORTRAN Syntax

```
INTEGER function gsfci (cx, cy, cr)
```

```
INTEGER cx, cy, cr
```

Pascal Syntax

```
FUNCTION gsfci_ (
```

```
  VAR cx, cy, cr : INTEGER  
): INTEGER [PUBLIC];
```

Description

The **gsfci** subroutine fills a circle of the specified radius. The radius is expressed in number of pixels.

The relevant attributes are:

- Color map
- Plane mask
- Fill color index
- Fill pattern index
- Logical operation.

Parameters

cx, cy Define the coordinates of the center of the circle.

cr Defines the radius of the circle.

If the radius is zero, a single point is filled at the center.

The fill pattern does not meet the border of the filled area on printers and plotters.

Return Value

GS_SUCC	Successful.
GS_CORD	Invalid coordinate.
GS_RDUS	Invalid radius specification.
GS_INAC	Virtual terminal inactive.

gsfell

gsfell

Purpose

Fills an ellipse.

C Syntax

```
int gsfell_ (cx, cy, ma, mi, ang)
```

```
int *cx, *cy, *ma, *mi, *ang;
```

FORTRAN Syntax

```
INTEGER function gsfell (cx, cy, ma, mi, ang)
```

```
INTEGER cx, cy, ma, mi, ang
```

Pascal Syntax

```
FUNCTION gsfell_ (
```

```
  VAR cx, cy, ma, mi, ang : INTEGER  
): INTEGER [PUBLIC];
```

Description

The **gsfell** subroutine fills an ellipse of the specified axes and angle. The axes are expressed in number of pixels.

The angle specifications are given in tenths of degrees, from 0 to 3600. Values outside this range cause the **gsfell** subroutine to fail.

The relevant attributes are:

- Color map
- Plane mask
- Fill color index
- Fill pattern index
- Logical operation.

Parameters

<i>cx, cy</i>	Define the coordinates of the center of the ellipse.
<i>ma, mi</i>	Define half of the nonzero major and minor axes of the ellipse.
<i>ang</i>	Defines the angle between the major axis and the x-axis. If it is zero, the major axis is on the x-axis and the minor axis is on the y-axis. The angle is defined in tenths of degrees, from 0 to 3600, specified in a counterclockwise direction.

The fill pattern does not meet the border of the filled area on printers and plotters.

Return Value

GS_SUCC	Successful.
GS_CORD	Invalid coordinate.
GS_ELMM	Invalid major or minor axis.
GS_INAC	Virtual terminal inactive.
GS_ANGL	Invalid angle.
GS_NMEM	Insufficient resources.

gsfply

gsfply

Purpose

Draws a filled polygon.

C Syntax

```
int gsfply_ (number, x, y)
```

```
int *number, *x, *y;
```

FORTRAN Syntax

```
INTEGER function gsfply (number, x, y)
```

```
INTEGER number
```

```
INTEGER x (*)
```

```
INTEGER y (*)
```

Pascal Syntax

```
FUNCTION gsfply_ (
```

```
VAR number: INTEGER;
```

```
VAR x, y: ARRAY [1..k] of INTEGER
```

```
): INTEGER [PUBLIC];
```

Description

The **gsfply** subroutine fills an area that is described by the points defined in the *number* and *x*, *y* parameters, with the color determined by the last call to the **gsfatt** subroutine.

The relevant attributes are:

- Color map
- Plane mask
- Fill color index
- Fill pattern index
- Logical operation.

Parameters

number Defines the number of points in the coordinate arrays. This value must be 3 or more.

x, y Define, as coordinate arrays, the points surrounding the polygon to fill.

The edges are treated as part of the area to be filled.

The **gsfpoly** subroutine fills a closed polygon with a pattern, generated by creating an edge between the first and the last points. The first and the last points described by the parameters may be equal, but it is not required and is actually less efficient.

The fill pattern does not meet the border of the filled area on printers and plotters.

For Pascal, the application must declare the arrays passed as being fixed length and declare the routine as accepting arrays of that length; that is, the *k* in the routine declaration must be a constant.

Return Value

GS_SUCC	Successful.
GS_CORD	Invalid coordinate.
GS_NCOR	Invalid number of coordinates.
GS_NMEM	Insufficient resources.
GS_INAC	Virtual terminal inactive.

gsfrec

gsfrec

Purpose

Draws a filled rectangle.

C Syntax

```
int gsfrec_ (x1, y1, x2, y2)
```

```
int *x1, *y1, *x2, *y2;
```

FORTRAN Syntax

```
INTEGER function gsfrec (x1, y1, x2, y2)
```

```
INTEGER x1, y1, x2, y2
```

Pascal Syntax

```
FUNCTION gsfrec_ (
```

```
  VAR x1, y1, x2, y2: INTEGER  
): INTEGER [PUBLIC];
```

Description

The **gsfrec** subroutine fills the rectangular area defined by the lower leftmost and upper rightmost coordinate parameters, with the color determined by the last call to the **gsfatt** subroutine.

The relevant attributes are:

- Color map
- Plane mask
- Fill color index
- Fill pattern index
- Logical operation.

Parameters

x1, y1 Define the lower left corner of the rectangular area to fill.

x2, y2 Define the upper right corner of the rectangular area to fill.

The edges of the rectangle are treated as part of the area to be filled.

The fill pattern does not meet the border of the filled area on printers and plotters.

Return Value

GS_SUCC	Successful.
GS_CORD	Invalid coordinate.
GS_INAC	Virtual terminal inactive.

gsgtat

gsgtat

Purpose

Sets the attributes for the geometric text drawing functions.

C Syntax

```
int gsgtat_ (color, baseline, pre, expan, spac, height,  
             upvectx, upvecty, alignhz, alignvt, font-ID, font)
```

```
int *color, *baseline, *pre, *expan, *spac, *height,  
int *upvectx, *upvecty, *alignhz, *alignvt, *font-ID;  
char *font;
```

FORTRAN Syntax

```
INTEGER function gsgtat (color, baseline, pre, expan, spac, height,  
                        upvectx, upvecty, alignhz, alignvt, font-ID, font)
```

```
INTEGER color, baseline, pre, expan, spac, height  
INTEGER upvectx, upvecty, alignhz, alignvt, font-ID  
CHARACTER*n font
```

Pascal Syntax

```
FUNCTION gsgtat_ (
```

```
  VAR color, baseline, pre, expan, spac, height: INTEGER;  
  VAR upvectx, upvecty, alignhz, alignvt, font-ID: INTEGER;  
  VAR font: ARRAY [0..k] of CHAR  
): INTEGER [PUBLIC];
```

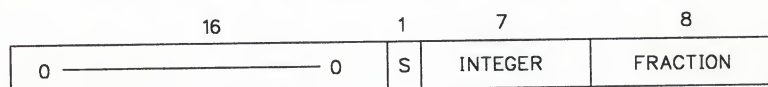

Description

The **gsgtat** subroutine defines the attributes and fonts for the geometric text drawing functions.

Note: The attributes defined by this command are applicable only to geometric text.

Parameters

- color** Specifies an entry in the color map for text color. If it is -1, the attribute is unchanged.
- baseline** Determines the direction of the geometric text drawing. The valid values are:
- 1 Attribute remains unchanged.
 - 0 Specifies 0 degrees, or left to right in the viewer's terms.
 - 1 Specifies 90 degrees, or up in the viewer's terms.
 - 2 Specifies 180 degrees, or right to left in the viewer's terms.
- Note:** The characters appear upside down.
- 3 Specifies 270 degrees, or down in the viewer's terms.
- Note:** The *baseline* parameter does not change character rotation. Use the *upvectx* and *upvecty* parameters to rotate text.
- pre** Specifies the desired text precision used in drawing text primitives. The valid values are:
- 1 Attribute remains unchanged.
 - 1 Character precision.
 - 2 Stroke precision.
- expan** Defines as a 32-bit fractional integer the deviation of the width/height ratio of the character from the ratio defined in the font. The expansion factor only changes the width of the character.



In the preceding figure, the first 16 bits contain zeros, S represents the sign bit, INTEGER represents the integer portion of the width/height ratio, and FRACTION represents the fractional portion of the ratio.

	A 32-bit integer value of 0x8000000 indicates that this attribute is unchanged.
<i>spac</i>	Specifies the character spacing, or additional number of pixels to be inserted between characters. The value is a 16-bit signed integer. The preferred value for this parameter varies, based on the display in use. The maximum value that is allowed is equal to the display width in pixels. A value of 0x8000000 for this parameter indicates that the attribute is unchanged.
<i>height</i>	Specifies the current character height for geometric text in pixels. This value is defined as a 16-bit signed integer, with the maximum value equal to the height of the display in pixels. A value of 0x8000000 for this parameter indicates that the attribute is unchanged.
<i>upvectx, upvecty</i>	Specify the x and y coordinates for the up direction of a character or text string. The valid range for these values is \pm the display dimensions in pixels. A value of 0x8000000 for this parameter indicates that the attribute is unchanged. The up vector is a two-dimensional vector on the text plane, specified by the current text draw. (The origin of the vector is defined by the geometric text command, gsgtxt .) Only the direction, not the length, of the vector is relevant.
<i>alignhz</i>	Specifies the horizontal alignment of the text for subsequent text drawing. Values are as follows: -1 Attribute is unchanged 1 Normal 2 Left 3 Center 4 Right.
<i>alignvt</i>	Specifies the vertical alignment of the text for subsequent text drawing. Values are as follows: -1 Attribute is unchanged 1 Normal 2 Top 3 Cap 4 Half 5 Base 6 Bottom.

font-ID

Specifies the ID of the font as a 32-bit integer, which defines the type of font to use. This ID is determined by the user while defining each geometric font. Possible values are:

- 1 A **font-ID** has been defined in a previous call to the **gsgtat** subroutine, and this attribute is unchanged.
- 1025 to 32767 These values are used to specify 1-byte geometric fonts, and refer to a value defined in each geometric font file.
- 32768 to 65535 These values are used to specify 2-byte geometric fonts, and refer to a value defined in each geometric font file.

Only 1 *font-ID* is active at any time. To change the **font-ID**, **gsgtat** must be called again with new *font-ID* and *font* parameters. When a new *font-ID* is specified, the previous *font-ID* is purged from the font table.

For 2-byte geometric text, up to 128 segment IDs can be used per *font-ID*.

When used with the *font* parameter, the *font-ID* is associated with the *font* used for font selection.

font

Contains the null-terminated full path name of the file used when the font attribute is specified as user. If a **font-ID** is defined, this parameter must also be defined. A value of -1 for this parameter indicates that the attribute is unchanged. For information on the format of font files for geometric text, see "Geometric Text Font Format" on page 3-88.

Attributes are only valid for the currently active font.

This subroutine must be called before the **gsgtxt** subroutine or an error results.

For Pascal, the application must declare the arrays passed as being fixed length and declare the routine as accepting arrays of that length. The *k* in the routine declaration must be a constant.

Return Value

GS_SUCC	Successful.
GS_COLI	Invalid color index.
GS_PREC	Invalid text precision value.
GS_EXPN	Invalid character expansion factor.
GS_FNTN	Invalid file name.
GS_INSV	Invalid spacing value.
GS_BASL	Invalid baseline direction.
GS_HIGH	Invalid height value.
GS_UPVT	Invalid up vector value.
GS_ALGN	Invalid alignment value.

Related Information

In this book: “fonts” on page 3-79, “Geometric Text Font Format” on page 3-88, and “gsgtxt” on page 6-89.

gsgtxt

Purpose

Writes geometric text.

C Syntax

```
int gsgtxt( x, y, number, text )
```

```
int *x, *y, *number;  
char *text;
```

FORTRAN Syntax

```
INTEGER function gsgtxt ( x, y, number, text )
```

```
INTEGER x, y, number  
CHARACTER*n text
```

Pascal Syntax

```
FUNCTION gsgtxt (
```

```
VAR x, y, number: INTEGER;  
VAR text: ARRAY [1..k] of CHAR  
): INTEGER [PUBLIC];
```

Description

The **gsgtxt** subroutine writes geometric characters starting at the baseline position defined by the parameters and writes the number of characters indicated by the parameters according to the relevant attributes.

The relevant attributes are:

- Color map
- Plane mask
- Font
- Text color index
- Character expansion factor

gsgtxt

- Character spacing
- Character height
- Character up vector
- Character alignment
- Baseline direction.

Parameters

<i>x, y</i>	Define the coordinates of the baseline position for writing geometric text.
<i>number</i>	Indicates the number of bytes to write from the <i>text</i> string. The maximum number of characters allowed is 1024 for single byte fonts and 512 for 2-byte fonts, which is determined by the display and font in use.
<i>text</i>	Contains the N-bit ASCII codes for the characters to write, as an array.

Return Value

GS_SUCC	Successful.
GS_CORD	Invalid coordinate.
GS_FBUF	Frame buffer overflow.
GS_INAC	Virtual terminal inactive.
GS_NOFT	Font not loaded.

Related Information

In this book: “fonts” on page 3-79, “Geometric Text Font Format” on page 3-88, “gsgtat” on page 6-84, and “gsqgtx” on page 6-138.

gsinit

Purpose

Initializes the GSL subroutines.

C Syntax

```
int gsinit_ (buffer, size, save_restore, f_grant, f_retract, fildes)
```

```
int *buffer, *size, *save_restore;  
int (*f_grant) ( ), (*f_retract) ( );  
int *fildes;
```

FORTRAN Syntax

```
INTEGER function gsinit (buffer, size, save_restore, f_grant, f_retract, fildes)
```

```
INTEGER buffer (*), size, save_restore, fildes  
EXTERNAL f_grant, f_retract
```

Pascal Syntax

```
FUNCTION gsinit_ (
```

```
VAR buffer: ARRAY [0..k] of INTEGER;  
VAR size, save_restore, f_grant, f_retract, fildes: INTEGER  
): INTEGER [PUBLIC];
```

Description

The **gsinit** subroutine initializes the GSL. It allocates any private storage required, and sets attributes to the default values where necessary. It also forces the virtual terminal of the application to monitor mode and sets up the signal processing routines for the **SIGRETRACT** and **SIGGRANT** signals, and optionally, the **SIGMSG** signal.

Parameters

<i>buffer</i>	Defines the monitor mode input ring buffer to be used by the GSL input functions. <i>buffer</i> must be word-aligned and at least 128 bytes long. For output to a printer or plotter device, set the <i>buffer</i> parameter to -1. (In C, <i>buffer</i> is a pointer to an integer containing the value -1. In Pascal, it is a variable containing the value -1.)						
<i>size</i>	Defines the length of <i>buffer</i> in bytes. Depending on the value of <i>size</i> , gsinit performs the following actions: <table><tr><td><i>size</i> = 0</td><td>The GSL ignores the <i>buffer</i> parameter and does not provide input support. The application must provide a means for receiving input events and can use the read system call or set up its own ring buffer mechanism. The IBM 5081 Display Adapter requires a ring buffer for input events. If you do not define a buffer (that is, if <i>size</i> = 0 and <i>buffer</i> is not defined), the GSL defines a buffer to be used only by GSL for the IBM 5081 Display Adapter. If you define a ring buffer after this point, the IBM 5081 Display Adapter GSL will not work.</td></tr><tr><td><i>size</i> < 128</td><td>The gsinit subroutine does not initialize the GSL.</td></tr><tr><td><i>size</i> ≥ 128</td><td>The GSL establishes the virtual terminal linkage to the input ring buffer provided by the application and provides input support and sets up a SIGMSG signal catcher.</td></tr></table>	<i>size</i> = 0	The GSL ignores the <i>buffer</i> parameter and does not provide input support. The application must provide a means for receiving input events and can use the read system call or set up its own ring buffer mechanism. The IBM 5081 Display Adapter requires a ring buffer for input events. If you do not define a buffer (that is, if <i>size</i> = 0 and <i>buffer</i> is not defined), the GSL defines a buffer to be used only by GSL for the IBM 5081 Display Adapter. If you define a ring buffer after this point, the IBM 5081 Display Adapter GSL will not work.	<i>size</i> < 128	The gsinit subroutine does not initialize the GSL.	<i>size</i> ≥ 128	The GSL establishes the virtual terminal linkage to the input ring buffer provided by the application and provides input support and sets up a SIGMSG signal catcher.
<i>size</i> = 0	The GSL ignores the <i>buffer</i> parameter and does not provide input support. The application must provide a means for receiving input events and can use the read system call or set up its own ring buffer mechanism. The IBM 5081 Display Adapter requires a ring buffer for input events. If you do not define a buffer (that is, if <i>size</i> = 0 and <i>buffer</i> is not defined), the GSL defines a buffer to be used only by GSL for the IBM 5081 Display Adapter. If you define a ring buffer after this point, the IBM 5081 Display Adapter GSL will not work.						
<i>size</i> < 128	The gsinit subroutine does not initialize the GSL.						
<i>size</i> ≥ 128	The GSL establishes the virtual terminal linkage to the input ring buffer provided by the application and provides input support and sets up a SIGMSG signal catcher.						
<i>save_restore</i>	Determines whether to save the display frame buffer and adapter states. If <i>save_restore</i> is nonzero, the GSL saves the current contents of the display frame buffer as well as the current adapter state when the virtual terminal must become inactive and restores both the frame buffer contents and adapter state when it becomes active. If <i>save_restore</i> is zero, the GSL saves only the adapter state and assumes that the application either saves the frame buffer or reconstructs it in some fashion.						
<i>f_grant</i>	Sets up processing of the SIGGRANT signal. If <i>f_grant</i> is nonzero, it is assumed to be the address of an application-supplied function, and the GSL calls the function as part of the SIGGRANT signal handling. If <i>save_restore</i> is nonzero, the application function is called before the frame buffer is restored.						
<i>f_retract</i>	Sets up processing of the SIGRETRACT signal. If <i>f_retract</i> is nonzero, it is assumed to be the address of an application-supplied function, and the GSL calls the function as part of the SIGRETRACT signal handling.						

fildev

Determines where output is directed. The output device is specified by one of the following:

- The value -1, which specifies standard output.
- A file descriptor returned by a **creat**, **open**, **dup**, or **fcntl** system call.
- A null-terminated character string up to 11 characters long, which names an environment variable defining a printer or plotter device. In this case, the value of the *buffer* parameter must be -1. (See "Printers and Plotters" on page 6-21.)

(In C, *fildev* is a pointer to a file descriptor, an integer, or a character string. In Pascal, it is a variable containing one of these values.)

If the initialization process is unsuccessful, the virtual terminal is not placed in monitor mode and invocation of any other GSL routines will cause unpredictable results.

For printers or plotters, if initialization is unsuccessful, the application can either terminate or re-drive the initialize function with a valid character string as a means of correcting the problem.

For Pascal, the application must declare the arrays passed as being fixed length and declare the routine as accepting arrays of that length; that is, the *k* in the routine declaration must be a constant.

Pascal cannot directly provide the address of a routine. An assembler function can be used to derive the address of a routine passed to the GSL.

The *f_grant* and *f_retract* routines supplied by the application are called on the signal level and *must return*. These application routines must not use either **setjmp** or **longjmp** subroutines.

The GSL supports use of the **sdb** symbolic debugger by redirection to a supplied file descriptor. If two virtual terminals are open and the GSL application runs on one, the application may get the file descriptor for the second and supply that descriptor at GSL initialization. The GSL directs its output to the second virtual terminal while **sdb** directs its output to the first; either is activated in the standard manner.

The user routine called at **SIGGRANT** can be called before **gsinit** returns to the application.

Return Value

GS_SUCC	Successful.
GS_HBUS	Cannot access hardware bus.
GS_ADPT	Invalid display type.
GS_FONT	Cannot access default font.
GS_RING	Buffer too small.
GS_HDCP	Invalid file descriptor for hard-copy output.

gsinit

GS_HDLK	Unable to create lock file.
GS_HDIM	Insufficient memory.
GS_HDDB	Device is busy.
GS_HDNA	Physical device not attached.
GS_HDMG	Maximum number of graphics devices open.
GS_HDIF	No system interprocess communication buffers left.
GS_HDSF	The fork system call failed.
GS_HDGO	Specified graphics device already open.
GS_HDGN	Specified graphics device does not exist.
GS_HDGU	Specified graphics device driver is unknown.

Related Information

In this book: "Printers and Plotters" on page 6-21.

gslatt

Purpose

Sets the line attributes.

C Syntax

```
int gslatt_ (color, style)
```

```
int *color, *style;
```

FORTRAN Syntax

```
INTEGER function gslatt (color, style)
```

```
INTEGER color, style
```

Pascal Syntax

```
FUNCTION gslatt_ (
```

```
  VAR color, style: INTEGER  
): INTEGER [PUBLIC];
```

Description

The **gslatt** subroutine defines the attributes for the class of line drawing functions.

Parameters

color

Refers to a line color entry in the color map. If it is -1, the attribute is unchanged. The default color is 15.

gslatt

style Sets or resets the line style pattern. The line *style* may be one of the following:

Value	Display	Printer or Plotter
-1	No change	No change
0	Solid	Solid
1	Dash	Dash
2	Dot	Dot
3	Dash-dot	Dash-dot
4	Dash-dot-dot	Dash-dot-dot
100	Continuous solid	Solid
101	Continuous dash	Dash
102	Continuous dot	Dot
103	Continuous dash-dot	Dash-dot
104	Continuous dash-dot-dot	Dash-dot-dot
150	Continuous user-supplied	Not available

The default style is solid (0).

The GSL supplied line style patterns are implemented in a device-dependent fashion. All line style indices not described above are reserved.

For line styles 1-99, the GSL line drawing functions ensure that a line or line segment starts and ends with a run of the line color. For example, the GSL *does not* continue the pattern from one polyline segment to another.

For line styles 100-150, the GSL continues the pattern across multiple lines or line segments until the application makes another call to **gslatt** to reset the line pattern. In this case, unlike styles 1-99, the GSL *does* continue the pattern from one polyline segment to another. Continuous line styles are not available on printers and plotters.

Return Value

GS_SUCC	Successful.
GS_COLI	Invalid color index.
GS_SYLI	Invalid style index.

gslcat

Purpose

Sets the locator attributes.

C Syntax

```
int gslcat_ (hg, vg)
```

```
int *hg, *vg;
```

FORTRAN Syntax

```
INTEGER function gslcat (hg, vg)
```

```
INTEGER hg, vg
```

Pascal Syntax

```
FUNCTION gslcat_ (
```

```
  VAR hg, vg: INTEGER  
): INTEGER [PUBLIC];
```

Description

The **gslcat** subroutine sets the locator attributes. Its effect depends on the type of locator attached. For a mouse, **gslcat** sets the thresholds. For a tablet, it sets the dead zone.

Parameters

hg, vg Define the horizontal and vertical values for the locator threshold or dead zone, in units of 0.25 millimeter.

The mouse **thresholds** determine the granularity of input events reported, or the amount of horizontal or vertical mouse movement required before an event occurs.

The tablet **dead zone** is an area of the tablet in which no event reports occur, even if the tablet sensor is present. This dead zone allows the application to make the tablet aspect ratio compatible with the display and allows tablets of different sizes to appear the same size to an application. The dead zone acts as a border around the tablet. The device driver

gslcat

reports movement only when the *x* value is greater than or equal to *hg* or less than or equal to (maximum tablet value - *hg*), and the *y* value is greater than or equal to *vg* or less than or equal to (maximum tablet value - *vg*).

An attempt to set the locator attributes may fail for a variety of reasons, the most likely of which is that the device is not attached. The nature of the problem can be determined with a specific **ioctl** to the virtual terminal. (See “hft” on page 5-39 for more information.)

Note that the **gslcat** subroutine allows an application to set the mouse thresholds or the tablet dead zone such that no events occur even if the device is enabled.

Return Value

GS_SUCC	Successful.
GS_USUC	Unsuccessful.

Related Information

In this book: “hft” on page 5-39.

gsline

Purpose

Draws a line between two points.

C Syntax

```
int gsline- (x1, y1, x2, y2)
```

```
int *x1, *y1, *x2, *y2;
```

FORTRAN Syntax

```
INTEGER function gsline (x1, y1, x2, y2)
```

```
INTEGER x1, y1, x2, y2
```

Pascal Syntax

```
FUNCTION gsline- (
```

```
VAR x1, y1, x2, y2: INTEGER  
): INTEGER [PUBLIC];
```

Description

The **gsline** subroutine draws a line, as defined by the current relevant attributes, from the first point to the second point defined by the parameters.

The relevant attributes are:

- Color map
- Plane mask
- Line color index
- Line style
- Logical operation.

gsline

Parameters

<i>x1, y1</i>	Defines the coordinates of one end point of the line drawn by gsline .
<i>x2, y2</i>	Defines the coordinates of the second point of the line drawn by gsline .

Return Value

GS-SUCC	Successful.
GS-CORD	Invalid coordinate.
GS-INAC	Virtual terminal inactive.

gslock

Purpose

Postpones signal processing.

C Syntax

```
int gslock_ ( )
```

FORTRAN Syntax

```
INTEGER function gslock ( )
```

Pascal Syntax

```
FUNCTION gslock_ ( ): INTEGER [PUBLIC];
```

Description

The **gslock** subroutine causes the GSL not to acknowledge the **SIGRETRACT** signal, if it occurs, until the application requests resumption of the signal handling with the **gsunlk** subroutine. This permits the application to access the display frame buffer directly.

If the virtual terminal is inactive when the application calls **gslock** and the GSL has been instructed to save the frame buffer when the virtual terminal becomes inactive, **gslock** suspends the application until the virtual terminal becomes active and then returns a successful return code. If the GSL has been instructed not to save the frame buffer, **gslock** returns the **GS_INAC** return code immediately. The application must not access the display frame when **GS_INAC** is returned.

Note: If **SIGRETRACT** signal processing is suspended for more than 30 seconds, it is possible that a generated **SIGRETRACT** signal may be suspended long enough for the **SIGKILL** signal to occur, terminating the application process.

gslock

Return Value

GS_SUCC	Virtual terminal active; safe to write to frame buffer.
GS_INAC	Virtual terminal inactive.

gslop

Purpose

Specifies the logical operation used when drawing lines.

C Syntax

```
int gslop_ (operation)
```

```
int *operation;
```

FORTRAN Syntax

```
INTEGER function gslop (operation)
```

```
INTEGER operation
```

Pascal Syntax

```
FUNCTION gslop_ (
```

```
  VAR operation INTEGER;  
): INTEGER [PUBLIC];
```

Description

The **gslop** subroutine specifies the logical operation used for drawing the GSL line-oriented, fill, save/restore, and polymarker primitives. It does not apply to the text primitives.

Parameters

operation

Indicates the logical operation to perform between the primitive being drawn and the current contents of the frame buffer.

In the following table, please note:

- The source pixels represent bits of data to be merged in some way with the corresponding bits of data in the destination rectangle.

- The first three columns of the table specify the operations you can perform, and the **Code** column contains the corresponding value you should specify for the *operation* parameter.
- A ~ (tilde) represents the logical INVERSE.

Type of Source	Logical Operation	Type of Destination	Code
		Destination clear	0
		Set Destination	15
	No operation	Destination	5
		~Destination	10
Source	REPLACE	Destination	3
Source	AND	Destination	1
Source	AND	~Destination	2
Source	Exclusive-or	Destination	6
Source	OR	Destination	7
Source	OR	~Destination	11
~Source	REPLACE	Destination	12
~Source	AND	Destination	4
~Source	AND	~Destination	8
~Source	Exclusive-or	Destination	9
~Source	OR	Destination	13
~Source	OR	~Destination	14

Replace (3) is the default logical operation.

Currently, the GSL provides only replace and exclusive-or (codes 3 and 6 respectively) for displays other than the IBM 5081 Display. The full set of logical operations is supported for the IBM 5081 Display.

For printers and plotters, the operations performed are the same as those for displays, except that a value of 0 turns the color off, and a value of 15 changes the color to white.

The GSL performs each of the Boolean operations for each bit of the source and destination color values enabled by the plane mask. The destination receives the color value that results from the operation.

The logical operations are performed on the color index rather than the color itself. This can cause some operations on color displays to produce results that are not expected.

Return Value

GS_SUCC
GS_LONS

Successful.
Logical operation not supported.

gslpat

gslpat

Purpose

Sets the Lighted Program Function Keyboard (LPFK) indicators.

C Syntax

```
int gslpat_ (indicators)
```

```
int *indicators;
```

FORTRAN Syntax

```
INTEGER function gslpat (indicators)
```

```
INTEGER indicators
```

Pascal Syntax

```
FUNCTION gslpat_ (
```

```
  VAR indicators: INTEGER  
): INTEGER [PUBLIC];
```

Description

The **gslpat** subroutine turns on or off the indicators on the Lighted Program Function Keyboard.

Parameters

indicators Specifies the state of the LPFK indicators. Each bit of *indicators* corresponds to an indicator on the LPFK, with the most significant bit setting the desired state (1 = on, 0 = off) for the indicator for LPFK 0, the next most significant bit setting the state for the indicator for LPFK 1, and so on.

The default state for all indicators is off.

An attempt to set the LPFK indicators can fail for a variety of reasons, the most likely of which is that the device is not attached. The nature of the problem can be determined with a specific **ioctl** to the virtual terminal. (See “hft” on page 5-39 for more information.)

Return Value

GS_SUCC	Successful.
GS_USUC	Unsuccessful.

Related Information

In this book: “hft” on page 5-39.

gsmask

gsmask

Purpose

Defines planes to be modified.

C Syntax

```
int gsmask_ (mask)
```

```
int *mask;
```

FORTRAN Syntax

```
INTEGER function gsmask (mask)
```

```
INTEGER mask
```

Pascal Syntax

```
FUNCTION gsmask_ (
```

```
  VAR mask: INTEGER  
): INTEGER [PUBLIC];
```

Description

The **gsmask** subroutine defines the planes actually modified by the line, text, and fill functions.

Parameters

<i>mask</i>	Indicates which planes of the display adapter frame buffer can be modified by the output functions. The most significant bits of the input are used to set the plane mask.
-------------	--

Return Value

GS_SUCC
GS_INAC

Successful.
Virtual terminal inactive.

gsmatt

gsmatt

Purpose

Sets the polymarker attribute.

C Syntax

```
int gsmatt_ (color, style, width, height, pattern, 0x, 0y)
```

```
int *color, *style, *width, *height, *pattern, *0x, *0y;
```

FORTRAN Syntax

```
INTEGER function gsmatt (color, style, width, height, pattern, 0x, 0y)
```

```
INTEGER color, width, height, pattern, 0x, 0y
```

Pascal Syntax

```
FUNCTION gsmatt_ (
```

```
  VAR color, style, width, height: INTEGER;
```

```
  pattern: ARRAY [1..k] of INTEGER;
```

```
  0x, 0y: INTEGER
```

```
): INTEGER [PUBLIC];
```

Description

The **gsmatt** subroutine defines the marker for the GSL.

Parameters

color Refers to a marker color entry in the color map. If it is -1, the attribute is unchanged. The default value for *color* is 7, white.

style Defines the polymarker *style* as one of the following:

Value	Display	Printer or Plotter
-1	No change	No Change
0	User-defined (by <i>width</i> , <i>height</i> , <i>pattern</i> , <i>0x</i> , <i>0y</i>)	Not available
1	Dot (filled circle)	Point
2	Plus (+)	Plus (+)
3	Asterisk (*)	Asterisk (*)
4	Circular shape	Square shape
5	Cross (x)	Cross (x)
6	Unfilled box	Diamond

width, *height* Define in pixels the width and the height of the bit pattern to be used as the marker. If *width* or *height* equals -1, then the pattern remains unchanged.

pattern Defines the image used as a marker. The ceiling of (*width* / 32) indicates the number of words per row and *height* indicates the number of rows. The marker data must be supplied in row (scan line) major order. If *width* implies partial use of a word, the rest of the word is unused. To fully define the marker pattern, *pattern* should be ((ceiling(*width* / 32)) × *height*) words in length.

0x, *0y* Indicate the coordinates of the origin of the marker relative to the lower leftmost corner (0, 0) of the marker pattern. The origin must be placed inside the marker pattern, so that *0x* < *width* and *0y* < *height*. The origin of the marker is placed at the position indicated when the application places a marker with the **gsplym** subroutine. (See "gsplym" on page 6-127.) If *0x* equals -1, then the origin remains unchanged.

The maximum size of the marker is device dependent. It equals the height and width of the display, which may be determined by calling the **gsqdsp** subroutine.

Note: The GSL subroutines do not make a copy of a user-defined polymarker. Changes or reuse of the storage where a user-defined shape is in use can cause unpredictable results.

For Pascal, the application must declare the arrays passed as being fixed length and declare the routine as accepting arrays of that length. The *k* in the routine declaration must be a constant.

Return Value

GS_SUCC	Successful.
GS_COLI	Invalid color index.
GS_PMSZ	Marker size invalid.
GS_PMOR	Marker origin invalid.
GS_PMSY	Marker style invalid.

Related Information

In this book: “gsplym” on page 6-127.

gsmcat

Purpose

Sets the cursor attributes.

C Syntax

int gsmcat_ (*foreground, background, width, height, pattern, mask, 0x, 0y, logop*)

int **foreground, *background, *width, *height, *pattern, *mask, *0x, *0y, *logop*)

FORTRAN Syntax

INTEGER function gsmcat_ (*foreground, background, width, height, pattern, mask, 0x, 0y, logop*)

INTEGER *foreground, background, width, height, pattern, mask, 0x, 0y, logop*)

Pascal Syntax

FUNCTION gsmcat_ (

VAR *foreground, background, width, height*: **INTEGER**;

VAR *pattern*: **ARRAY** [1..*k*] **of** **INTEGER**;

VAR *mask*: **ARRAY** [1..*k*] **of** **INTEGER**;

VAR *0x, 0y, logop*: **INTEGER**

): INTEGER [PUBLIC];

Description

The **gsmcat** subroutine defines and enables the multicolor cursor for the GSL. The **gsmmap** subroutine must initialize the color map before **gsmcat** can be called.

Only one cursor of the raster-style cursor, either the multicolor cursor or the single-color cursor, can be active in the GSL at any one time. The **gsmcat** subroutine forces all subsequent calls to the **gsmcur** and **gsecur** subroutines to operate on the multicolor version of the raster cursor. To change from the single-color cursor to the multicolor cursor, erase the cursor with **gsecur**, then call the **gsmcat** subroutine.

The multicolor cursor is a two-color, clipped cursor with logical operations. Its size is limited to 32 bits in width and 32 bits in height. Although the cursor origin cannot be

moved outside the frame buffer boundaries, any portion beyond the origin that falls outside the frame buffer is clipped. In addition, a mask is provided that can be used to allow portions of the frame buffer to *show through* the cursor. Any bits set to 0 in the mask indicate that the matching bits in the cursor pattern do not affect the underlying frame buffer.

Parameters

<i>foreground</i>	Defines a color entry in the color map. This color is used for the foreground color (bits set to 1) in the multicolor cursor raster. A value of -1 indicates no change to this attribute.				
<i>background</i>	Defines a color entry in the color map. This color is used for the background color (bits set to 0) in the multicolor cursor raster. A value of -1 indicates no change to this attribute.				
<i>width, height</i>	Define, in pixels, the width and height of the bit pattern and mask to be used as the cursor. The maximum value for width and height of the cursor is 32 bits. If <i>width</i> or <i>height</i> equals -1, then the pattern and the mask remain unchanged.				
<i>pattern</i>	Defines the raster image used as a cursor. It must be specified in 32-bit integers, and there must be <i>height</i> number of rows. The GSL will only use <i>width</i> number of bits in each integer.				
<i>mask</i>	Defines the mask pattern of the cursor. Each bit in the <i>mask</i> corresponds with a bit in the multicolor cursor <i>pattern</i> . If a bit is set (has a value of 1), the matching bit in the pattern is applied to the underlying display raster. If a bit is not set (has a value of 0), the matching bit in the pattern is <i>masked</i> and does not affect the underlying display raster. The size of the <i>mask</i> must match the size of the <i>pattern</i> exactly.				
<i>0x, 0y</i>	Indicate the origin of the cursor relative to the lower leftmost corner (0, 0) of the cursor pattern. The origin must be placed within the cursor pattern: $0x < width$ and $0y < height$. The origin of the cursor is placed at the position indicated, when the application moves the cursor using the <i>gsmcur</i> subroutine. If <i>x</i> equals -1, then the origin remains unchanged.				
<i>logop</i>	Defines the logical operation to perform between the cursor pattern being drawn and the contents of the frame buffer. The following logical operations are supported: <table><tr><td>3</td><td>REPLACE</td></tr><tr><td>6</td><td>Exclusive-OR</td></tr></table>	3	REPLACE	6	Exclusive-OR
3	REPLACE				
6	Exclusive-OR				

You cannot change the cursor attributes while the cursor is visible.

There is no default cursor defined, so all cursor parameters must be set before the cursor is displayed.

For Pascal, the application must declare the array passed as being fixed length and declare the routine as accepting arrays of that length. The *k* in the routine declaration must be a constant.

Return Value

GS_SUCC	Successful.
GS_COLI	Invalid color index.
GS_CURS	Cursor size invalid.
GS_CURO	Cursor origin invalid.
GS_CURV	Cursor visible.
GS_LONS	Invalid logical operation.

Related Information

In this book: "gscatt" on page 6-32, "gsecur" on page 6-56, and "gsmcur" on page 6-116.

gsmcur

gsmcur

Purpose

Moves the cursor and makes it visible.

C Syntax

```
int gsmcur_ (x, y)
```

```
int *x, *y;
```

FORTRAN Syntax

```
INTEGER function gsmcur (x, y)
```

```
INTEGER x, y
```

Pascal Syntax

```
FUNCTION gsmcur_ (
```

```
VAR x, y: INTEGER  
): INTEGER [PUBLIC];
```

Description

The **gsmcur** subroutine makes the cursor visible (if not already visible) and positions the cursor origin at the point indicated by the parameters. This subroutine operates on the single-color cursor, and the multicolor cursor. The relevant attributes are different, depending on the raster cursor style currently defined and enabled.

For the single-color cursor, the relevant attributes are:

- Color map
- Plane mask
- Cursor pattern
- Cursor color index
- Cursor origin.

For the multicolor cursor, the relevant attributes are:

- Color map
- Plane mask
- Multicolor cursor pattern
- Multicolor cursor mask
- Multicolor cursor foreground color
- Multicolor cursor background color
- Multicolor cursor origin
- Multicolor cursor logical operation.

Parameters

x, y Indicate the coordinates of the desired position of the cursor origin.

The cursor attributes must be set with the **gscatt** or **gsmcat** subroutine before calling **gsmcur**. (See “gscatt” on page 6-32 and “gsmcat” on page 6-113.)

The cursor is non-destructive. This is achieved in a device-dependent manner.

Return Value

GS_SUCC	Successful.
GS_CORD	Invalid coordinate.
GS_UCUR	Undefined cursor.
GS_INAC	Virtual terminal inactive.

Related Information

In this book: “gscatt” on page 6-32 , “gsecur” on page 6-56, and “gsmcat” on page 6-113

gsmfld

gsmfld

Purpose

Loads multiple fonts simultaneously.

C Syntax

```
int gsmfld_ (number-of-fonts, font-table)
```

```
int *number-of-fonts;  
struct font-info font-table[];
```

FORTRAN Syntax

```
INTEGER gsmfld (number-of-fonts, font-table)
```

```
INTEGER number-of-fonts, font-table
```

Pascal Syntax

```
type
```

```
    font-info = record  
        font-id : integer;  
        font-path : integer;  
    end;  
    font-array [1..k] of font-info;
```

```
FUNCTION gsmfld_ (
```

```
    VAR number-of-fonts: integer;  
    VAR font-table : font-array ): integer[PUBLIC];
```

Description

The **gsmfld** routine allows multiple fonts to be loaded into memory. The fonts to be loaded are specified in a font table.

Parameters

number-of-fonts Specifies the number of elements contained in *font-table*. This value is the number of fonts to load simultaneously into memory.

font-table A pointer to the start of an array of elements with the following structure:

```
struct font-info
{
    int font-id;
    char *font-path;
}
```

The *font-table* should contain the number of elements specified by *number-of-fonts*. Each element contains either a font ID or the full path of a font. See "gstatt" on page 6-153 for a complete list of font IDs. RT fonts should be loaded by font ID and user-defined fonts should be loaded by path. If a user-defined font is loaded, the *font-id* must be 0.

Any combination of RT and user-defined fonts can be loaded into memory. No more than 128 fonts can be loaded simultaneously. The number of fonts must be greater than 0. Each font ID must either be 0, or one of the font IDs listed in **gstatt**.

The **gsmfld** routine should only be called after GSL initialization has completed successfully from a call to **gsinit**. The **gsmfld** routine can only be called once within a GSL initialization or termination sequence. Therefore, you should ensure that all desired fonts are specified in the font table before calling **gsmfld**.

After **gsmfld** has executed successfully, fonts can be changed by calling the **gstatt** routine. However, fonts loaded by font ID must be referred to by font ID when calling **gstatt**. Similarly, fonts loaded by path must be referred to by path when calling **gstatt**.

Note: Once **gsmfld** has executed successfully, you cannot use fonts that were not previously included in the font table.

The most frequently used font should be the first font listed in the font table. The **gsmfld** routine is used to load both local and remote fonts.

Note: When **gsmfld** is called by an application running on a IBM 5081 Display, the requested fonts are loaded into DMA, assuming adequate DMA space is available. As a result, accessing and changing fonts will be faster. However, because fonts are loaded into DMA, with a consequent reduction in the amount of available DMA memory space, the ability to load geometric (stroke) fonts and the performance of the **gsxblt** routine may be affected. With the exception of the IBM 5081 Display or unless the fonts are remote, the **gsmfld** routine does not

gsmfld

provide any performance enhancement. In the latter case, the fonts are loaded in local memory and can be accessed more quickly than remote fonts.

Before a FORTRAN program calls **gsmfld**, the program must properly initialize the font table as an integer array. The array contains two integers for each font to be loaded. The first integer contains the font ID; the second one contains the address of the font's path. In addition, each character string that specifies a font path must be terminated with a null (0) character.

The following sample FORTRAN program `f test` loads two fonts, first a user-defined and then an RT font, using **gsmfld**.

c The FORTRAN program ftest load 2 fonts using gsmfld_ () loads
c one user-defined font and one RT font.

```
program ftest
integer buffer (128)
integer count, size, len, savres, fildes, x, y, rc
character*25 a
character*80 msg
integer fontray(2,2)
external gsinit, gsxptr, gsmfld, gstat, gstat, gsterm
```

msg = 'This is a string used to demonstrate the call gsmfld_\0'
c define the path of the user-defined font; put a 0 at string end

```
a = '/yourpath/font1\0'
```

c perform GSL initialization

```
size = 0
savres = -1
fildes = -1
call gsinit (buffer,size,savres,0,0,fildes)
```

c define the number of fonts to be loaded

```
count = 2
```

c set the font ID to 0 for the user-defined font

```
fontray (1,1) = 0
```

c set the font ID for the RT font as desired

```
fontray (1,2) = 12
```

c put the address of the user-defined font into the array

```
call gsxptr (a,fontray (2,1))
```

c ensure that the character pointer for the RT font is NULL

```
fontray (2,2) = 0
```

c load the fonts

```
rc = gsmfld (count,fontray)
```

c display each of the 2 fonts multiple times

```
x = 50
y = 50
len = 53
count = 1
do 11 x = 50, 300, 20
    y = y + 20
    if (count .eq. 1) then
        rc = gstatt (savres,savres,savres,fontray (1,1),a)
    else if (count .eq. 2) then
        rc = gstatt (savres,savres,savres,fontray (1,2),fontray (2,2))
    endif
    rc = gstext (x,y,len,msg)
    count = count + 1
    if (count .eq. 3) then
        count = 1
    endif
    continue
c terminate GSL and exit
rc = gsterm ()
stop
end
```

For Pascal, the application must declare the array passed as being fixed length and declare the routine as accepting arrays of that length. The *k* in the routine declaration must be a constant. You must store the address of the start of each font in the *font_path* fields in the record **font_info**. Each path must be contained in a **char** array with C-style structure. A C-style structure has no string length, and a null character ends the string.

Return Value

GS-SUCC	Successful.
GS-USUC	Failed.
GS-BCNT	Invalid number of fonts was specified.
GS-MFLD	Mutiple fonts are already loaded.
GS-FNTI	Invalid font ID in font table.
GS-NMEM	Failed allocating memory.
GS-FNTN	Invalid font path in font table.
GS-INAC	Virtual terminal inactive.
GS-BRED	Failed reading font file.
GS-NDMA	Not enough DMA memory space available for the specified fonts.

gsmult

Purpose

Draws a multiline, a set of lines that connect alternate pairs of points in a sequence.

C Syntax

```
int gsmult_ (number, x, y)
```

```
int *number, *x, *y;
```

FORTRAN Syntax

```
INTEGER function gsmult (number, x, y)
```

```
INTEGER number, x (*), y (*)
```

Pascal Syntax

```
FUNCTION gsmult_ (
```

```
  VAR number: INTEGER;  
  VAR x, y: ARRAY [1..k] of INTEGER  
): INTEGER [PUBLIC];
```

Description

The **gsmult** subroutine draws lines, as defined by the current relevant attributes, between alternate pair of points defined by the parameters.

The relevant attributes are:

- Color map
- Plane mask
- Line color index
- Line style
- Logical operation.

gsmult

Parameters

- number* Defines the number of points in the coordinate arrays. It must be a multiple of 2, with 2 as the minimum value.
- x, y* Define the points for line drawing.

For Pascal, the application must declare the arrays passed as being fixed length and declare the routine as accepting arrays of that length. The *k* in the routine declaration must be a constant.

Return Value

- | | |
|----------------|--------------------------------|
| GS-SUCC | Successful. |
| GS-CORD | Invalid coordinate. |
| GS-NCOR | Invalid number of coordinates. |
| GS-INAC | Virtual terminal inactive. |

gspcls

Purpose

Defines the end of a shape to fill.

C Syntax

`int gspcls_ ()`

FORTRAN Syntax

`INTEGER function gspcls`

Pascal Syntax

`FUNCTION gspcls_ : INTEGER [PUBLIC];`

Description

The **gspcls** subroutine defines the end of a particular two dimensional shape to be filled, then fills the shape.

See “gsbply” on page 6-28 and “gseply” on page 6-61 for related information.

The relevant attributes are:

- Color map
- Plane mask
- Fill color index
- Fill style
- Logical operation.

gspcls

Return Value

GS_SUCC	Successful.
GS_USUC	Unsuccessful.

Related Information

In this book: “gsbply” on page 6-28 and “gseply” on page 6-61.

gsplym

Purpose

Draws a polymarker, a marker at each of a set of specified points.

C Syntax

```
int gsplym- (number, x, y)
```

```
int *number, *x, *y;
```

FORTRAN Syntax

```
INTEGER function gsplym (number, x, y)
```

```
INTEGER number, x (*), y (*)
```

Pascal Syntax

```
FUNCTION gsplym- (
```

```
  VAR number: INTEGER;
```

```
  VAR x, y: ARRAY [1..k] of INTEGER
```

```
); INTEGER [PUBLIC];
```

Description

The **gsplym** subroutine places a marker, defined by the current relevant attributes, at each point defined by the parameters.

The relevant attributes are:

- Color map
- Plane mask
- Logical operation
(except on IBM 5081 Display)
- Polymarker color index
- Polymarker style index.

gsplym

Parameters

<i>number</i>	Defines the number of points in the coordinate arrays. It must be ≥ 1 .
<i>x, y</i>	Define, as coordinate arrays, the location where the origin of each polymarker is placed.

For Pascal, the application must declare the arrays passed as being fixed length and declare the routine as accepting arrays of that length. The *k* in the routine declaration must be a constant.

Return Value

GS-SUCC	Successful.
GS-CORD	Invalid coordinate.
GS-NCOR	Invalid number of coordinates.

gspoly

Purpose

Draws a polyline, a set of lines that connects a sequence of points.

C Syntax

```
int gspoly_ (number, x, y)
```

```
int *number, *x, *y;
```

FORTRAN Syntax

```
INTEGER function gspoly (number, x, y)
```

```
INTEGER number, x (*), y (*)
```

Pascal Syntax

```
FUNCTION gspoly_ (
```

```
  VAR number: INTEGER;
```

```
  VAR x, y: ARRAY [1..k] of INTEGER  
); INTEGER [PUBLIC];
```

Description

The **gspoly** subroutine draws lines, as defined by the current relevant attributes, between each pair of points defined by the parameters.

The relevant attributes are:

- Color map
- Plane mask
- Line color index
- Line style
- Logical operation.

Parameters

number Defines the number of points in the coordinate arrays. It must be ≥ 2 .

x, y Define the points for line drawing.

For Pascal, the application must declare the arrays passed as being fixed length and declare the routine as accepting arrays of that length. The *k* in the routine declaration must be a constant.

Return Value

GS-SUCC	Successful.
GS-CORD	Invalid coordinate.
GS-NCOR	Invalid number of coordinates.
GS-INAC	Virtual terminal inactive.

gspp

Purpose

Sets plotter pen speed.

C Syntax

```
int gspp- (penspd)
```

```
int *penspd;
```

FORTRAN Syntax

```
INTEGER function gspp (penspd)
```

```
INTEGER penspd
```

Pascal Syntax

```
FUNCTION gspp- (
```

```
  VAR penspd: INTEGER;  
): INTEGER [PUBLIC];
```

Description

The **gspp** subroutine sets the plotter pen speed.

Parameters

penspd

Specifies the pen speed as a value from 0 to 100, giving a percentage of the maximum speed of the plotter. The initial pen speed is 100 percent.

Return Value

GS-SUCC	Successful.
GS-USUC	Invalid parameter value.

gsqdsp

Purpose

Returns characteristics of the display monitor and adapter.

C Syntax

```
void gsqdsp- (display)
```

```
int *display;
```

FORTRAN Syntax

```
subroutine gsqdsp (display)
```

```
INTEGER display (32)
```

Pascal Syntax

```
PROCEDURE gsqdsp- (
```

```
VAR display: ARRAY [1..32] of INTEGER  
): INTEGER [PUBLIC];
```

Description

The **gsqdsp** subroutine returns an array containing the display adapter and monitor characteristics.

Parameters

display Contains, on return, the relevant display/monitor characteristics. The following table describes the information in the array. Each entry is a word.

Entry Description (measure)

- 1 Display/monitor ID. For a printer or plotter, this value is " HC", right-justified in the word.
- 2 Displayed width of the frame buffer in pixels.
- 3 Displayed height of the frame buffer in pixels.
- 4 Physical width of display in millimeters.
- 5 Physical height of display in millimeters.
- 6 Number of bit planes or number of bits/pixel.
- 7 Adapter characteristic flags. (Bit 0 is the most significant bit.) Bits set these characteristics:
 - 0 Color or monochrome; 0 = color, 1 = monochrome
 - 1 By plane or by pixel; 0 = by plane, 1 = by pixel
(always 1 for printers and plotters).
 - 2 Software or hardware cursor; 0 = software, 1 = hardware
(always 0 for printers and plotters).
 - 3-31 Reserved bits.
- 8 Number of bits for Red digital-to-analog converter (always 2 for printers and plotters).
- 9 Number of bits for Green digital-to-analog converter (always 2 for printers and plotters).
- 10 Number of bits for Blue digital-to-analog converter (always 2 for printers and plotters).
- 11 Minimum cursor width (pixels)
(always 0 for printers and plotters).
- 12 Minimum cursor height (pixels)
(always 0 for printers and plotters).
- 13 Maximum cursor width (pixels)
(always 0 for printers and plotters).
- 14 Maximum cursor height (pixels)
(always 0 for printers and plotters).
- 15 Color table size. For printers and plotters, this specifies the number of colors.
- 16 Font class:
 - 1 Compressed (always 1 for printers and plotters).
 - 2 Uncompressed.
- 17 Logical operation capability.
 If the value is zero, the adapter supports all 16 two-operand logical operations and all 256 three-operand logical operations. If nonzero, the most significant bits represent the two-operand logical operations supported; bit 0 corresponds to logical operation 0, bit 1 to logical operation 1, and so on
(see "gslop" on page 6-103).

Entry Description (measure)

18-32 Reserved.

Information from this query can be used to scale application coordinates to those of the frame buffer.

Even if the adapter supports **no** logical operations, the results of the query indicate that the adapter supports replace and exclusive-or (logical operations 3 and 6, respectively). The GSL emulates the latter, if necessary.

The following display adapter IDs are valid:

0x0402	IBM 6153 Display Adapter
0x0405	IBM 6155 Display Adapter
0x0406	IBM 6154 Display Adapter
0x0408	IBM 5081 Display Adapter.

Related Information

In this book: "gslop" on page 6-103.

gsqfnt

gsqfnt

Purpose

Returns information about the current font.

C Syntax

```
void gsqfnt_ (font)
```

```
int *font;
```

FORTRAN Syntax

```
subroutine gsqfnt (font)
```

```
INTEGER font (32)
```

Pascal Syntax

```
PROCEDURE gsqfnt_ (
```

```
  VAR font: ARRAY [1..32] of INTEGER  
): INTEGER [PUBLIC];
```

Description

The **gsqfnt** subroutine returns information about the active font.

Parameters

font

Contains, on return, the characteristics of the current font. The following table describes the information in the array. Each entry is a word. Dimensions are in pixels and the origin is at the lower left corner of the character box.

Entry Description

- 1 Class: 1 = compressed; 2 = uncompressed IBM 5081 Display format (always 1 for printers and plotters).
- 2 Font ID.
- 3 Style.
- 4 Attribute flags:
bit 31 bold
bit 30 italic
bit 00 proportionally spaced.
(This entry always has all bits set to 0 for printers and plotters.)
- 5 Number of characters. For printers and plotters, this is the number of fonts \times 128.
- 6 Character baseline. For printers and plotters, no text alignment is allowed and this value is always -1.
- 7 Character capsline. For printers and plotters, no text alignment is allowed and this value is always -1.
- 8 Character width. For printers and plotters, the character width is given in pixels. For a proportionally spaced font, the width value represents the maximum width allowed.
- 9 Character height. For printers and plotters, the character height is given in pixels.
- 10 Underscore top line. For printers and plotters, underscoring is not available and this value is always -1.
- 11 Underscore bottom line. For printers and plotters, underscoring is not available and this value is always -1.
- 12-32 Reserved.

gsqgtx

gsqgtx

Purpose

Returns information about the current geometric font.

C Syntax

```
void gsqgtx- (font, select)
```

```
int *font, *select;
```

FORTRAN Syntax

```
subroutine gsqgtx (font, select)
```

```
INTEGER font (32), select
```

Pascal Syntax

```
PROCEDURE gsqgtx- (
```

```
VAR font: ARRAY [1..32] of INTEGER;
```

```
  select: INTEGER
```

```
): INTEGER [PUBLIC];
```

Description

The **gsqgtx** subroutine returns information about the active geometric font.

Parameters

font

Contains, on return, the characteristics of the selected PCS descriptor header. The following table describes the information in the array. Each entry is a word. Dimensions are in pixels and the origin is at the lower left corner of the character box.

Entry	Description
1	Font ID.
2	Segment ID.
3	0 = EBCDIC; 1 = ASCII.
4	Range of x (P).
5	Range of y (Q).
6	Starting character code. Range is 0x21 to 0xFE.
7	Last character code. Range is 0x21 to 0xFE.
8	Font baseline. Value in pixels in the y direction.
9	Font capline. Value in pixels in the x direction.
10	Default error code point.
11-32	Reserved.

select

Determines the type of query.

A value of -1 returns the following information in the *font* parameter buffer:

Word 1 Current active *font-ID*

Word 2 Number of PCS descriptor headers
(segments for 2-byte text) loaded at the time of the query.

A value other than -1 returns the PCS descriptor header associated with that number in the table. The first entry is 0, with a range of 0 to *n*.

Related Information

In this book: “fonts” on page 3-79, “Geometric Text Font Format” on page 3-88, “gsgtat” on page 6-84, and “gsgtxt” on page 6-89.

gsqlext

gsqlext

Purpose

Returns expanded information about the locator.

C Syntax

```
int gsqlext_ (results )
```

```
int results[16];
```

FORTRAN Syntax

```
INTEGER function gsqlext (results)
```

```
INTEGER results (16 )
```

Pascal Syntax

```
FUNCTION gsqlext_ (
```

```
VAR results: ARRAY[1..16] of INTEGER;  
); INTEGER [PUBLIC];
```

Description

The **gsqlext** subroutine returns an array containing expanded information about the locator device.

Parameters

<i>results</i>	Contains, on return, information about the type of locator, the resolution of the locator device, the maximum horizontal and vertical counts, and the current setting of the relative device thresholds or the absolute device dead zone values. The following table describes the information in the array.
----------------	--

Entry Description

- 0 Locator resolution in millimeters per 100 counts.
- 1 Indicates the locator device type. If the most significant bit is:
 - 0 The locator type is a mouse. When the locator is a mouse, the setting of the following bits is ignored.
 - 1 The locator type is a tablet. For a tablet, the next most significant 2 bits are:
 - 00 Sensor type is undefined or no sensor is attached.
 - 01 A stylus is attached.
 - 10 A four-button puck is attached.
- 2 Maximum horizontal count.
- 3 Maximum vertical count.
- 4 The horizontal locator threshold or dead zone in units of 0.25 millimeter.
- 5 The vertical locator threshold or dead zone in units of 0.25 millimeter.
- 6-15 Reserved.

An attempt to get the locator attributes can fail for a variety of reasons, the most likely of which is that the device is not attached. The nature of the problem can be found via a specific **ioctl** to the virtual terminal. (See “hft” on page 5-39 for more information.)

For Pascal, the application must declare the array passed as being fixed length and declare the routine as accepting an array of that length. The *k* in the routine declaration must be a constant.

Return Value

GS_SUCC	Successful.
GS_USUC	Unsuccessful.

Related Information

In this book: “hft” on page 5-39 and “gsqloc” on page 6-142.

gsqloc

gsqloc

Purpose

Returns information about the locator.

C Syntax

```
void gsqloc_ (loc-type, x-res, y-res, hg, vg)
```

```
int *loc-type, *x-res, *y-res, *hg, *vg;
```

FORTRAN Syntax

```
subroutine gsqloc (loc-type, resolution, hg, vg)
```

```
INTEGER loc-type, resolution, hg, vg
```

Pascal Syntax

```
PROCEDURE gsqloc_ (
```

```
  VAR loc-type, resolution, hg, vg: INTEGER  
): INTEGER [PUBLIC];
```

Description

The **gsqloc** subroutine returns the type of the locator, the resolution of the device, and the current setting of the relative device thresholds or the absolute device dead zone values (see “gslcat” on page 6-97).

Parameters

<i>loc-type</i>	Indicates the type of locator. If the most significant bit of <i>loc-type</i> is 0, the locator is a mouse. When the locator is a mouse, the setting of the following bits is ignored. When the most significant bit is 1, the locator type is a tablet.
-----------------	--

For a tablet, the next most significant 2 bits are:

- 00 Sensor type is undefined or no sensor is attached.
- 01 A stylus is attached.
- 10 A four-button puck is attached.

x-res, y-res Indicate the horizontal and vertical resolution of the device in millimeters per 100 counts.

hg, vg Define the horizontal and vertical values for the locator threshold or dead zone in units of 0.25 millimeters.
(See "gslcat" on page 6-97.)

An attempt to get the locator attributes can fail for a variety of reasons, the most likely of which is that the device is not attached. The nature of the problem can be found via a specific **ioctl** to the virtual terminal.

(See "hft" on page 5-39 for more information.)

Return Value

GS_SUCC	Successful.
GS_UNSC	Unsuccessful.

Related Information

In this book: "hft" on page 5-39 and "gslcat" on page 6-97.

gsrrst

gsrrst

Purpose

Restores a rectangular block.

C Syntax

```
int gsrrst_ (buffer, x1, y1, x2, y2)
```

```
int *buffer, *x1, *y1, *x2, *y2;
```

FORTRAN Syntax

```
INTEGER function gsrrst (buffer, x1, y1, x2, y2)
```

```
INTEGER buffer (*), x1, y1, x2, y2
```

Pascal Syntax

```
FUNCTION gsrrst_ (
```

```
  VAR buffer: ARRAY [1..k] of INTEGER;  
  VAR x1, y1, x2, y2: INTEGER  
): INTEGER [PUBLIC];
```

Description

The **gsrrst** subroutine restores a block of pixels saved to the frame buffer by the **gsrsav** subroutine. (See “gsrsav” on page 6-146.)

The relevant attributes are:

- Plane mask
- Logical operation.

Parameters

<i>buffer</i>	Indicates where gsrrst should restore the block of pixels from.
<i>x1, y1</i>	Define the coordinates of the lower left corner of the rectangular area to restore.
<i>x2, y2</i>	Define the coordinates of the upper-right corner of the rectangular area to restore.

The intended purpose of the **gsrsav** and **gsrrst** subroutines is efficient saving and restoring of pixel blocks displayed temporarily at a fixed location in the frame buffer. Because the GSL saves the frame buffer contents in a device-dependent fashion, it is generally not possible to use **gsrsav** and **gsrrst** to correctly move blocks of pixels from one position to another in a plane oriented adapter, nor is it possible for the application to manipulate the *buffer* without careful consideration of adapter characteristics, block size, and position of the block in the frame buffer.

For further information on moving and storing blocks of pixels, see “gsxblt” on page 6-164.

For Pascal, the application must declare the array passed as being fixed length and declare the routine as accepting an array of that length. The *k* in the routine declaration must be a constant.

Return Value

GS-SUCC	Successful.
GS-CORD	Invalid coordinate.
GS-INAC	Virtual terminal inactive.

Related Information

In this book: “gsrsav” on page 6-146 and “gsxblt” on page 6-164.

gsrsav

gsrsav

Purpose

Saves a rectangular block.

C Syntax

```
int gsrsav_ (buffer, x1, y1, x2, y2)
```

```
int *buffer, *x1, *y1, *x2, *y2;
```

FORTRAN Syntax

```
INTEGER function gsrsav (buffer, x1, y1, x2, y2)
```

```
INTEGER buffer (*), x1, y1, x2, y2
```

Pascal Syntax

```
FUNCTION gsrsav_ (
```

```
VAR buffer: ARRAY [1..k] of INTEGER;
```

```
VAR x1, y1, x2, y2: INTEGER
```

```
): INTEGER [PUBLIC];
```

Description

The **gsrsav** subroutine saves a block of pixels, defined by the input rectangle, in storage starting at the address indicated. This stored block can be restored with the **gsrrst** subroutine. (See "gsrrst" on page 6-144.)

The relevant attributes are:

- Plane mask
- Logical operation.

Parameters

buffer

Indicates where **gsrsav** should save the block of pixels.

The size of the *buffer* depends on the size of the rectangle and on the device organization. For devices organized by plane, the plane mask attribute determines the number of planes saved for each pixel. For devices organized by pixel, the entire pixel is always saved. For both organizations, the unit of access to the frame buffer also plays a role in calculating the size of the *buffer*. See “gscmap” on page 6-41 for details.

Note that the **gsrsav** subroutine does not check whether the *buffer* is too small to contain the pixel block. Serious consequences can result if the *buffer* is too small. However, a *buffer* size equal to

$$((y2 - y1 + 1) / 32 + 2) * (x2 - x1 + 1)$$

will hold all save images.

x1, y1

Define the lower left corner of the rectangular area to save. That is, *x1* is the greatest lower bound of the pixels saved in *x*.

x2, y2

Define the upper-right corner of the rectangular area to save. That is, *x2* is the least upper bound of the pixels saved in *x*.

The intended purpose of the **gsrsav** and **gsrrst** subroutines is efficient saving and restoring of pixel blocks displayed temporarily at a fixed location in the frame buffer. Because the GSL saves the frame buffer contents in a device-dependent fashion, it is generally not possible to correctly move blocks of pixels from one position to another in a plane-oriented adapter using **gsrsav** and **gsrrst**, nor is it possible to manipulate the *buffer* without careful consideration of adapter characteristics, block size, and position of the block in the frame buffer.

For further information on moving and storing blocks of pixels, see “gsxblt” on page 6-164.

For Pascal, the application must declare the array passed as being fixed length and declare the routine as accepting an array of that length. The *k* in the routine declaration must be a constant.

Return Value

GS_SUCC

Successful.

GS_CORD

Invalid coordinate.

GS_INAC

Virtual terminal inactive.

Related Information

In this book: “gscmap” on page 6-41, “gsrrst” on page 6-144, and “gsxblt” on page 6-164.

gssend

Purpose

Sends adapter level commands directly to the display adapter hardware.

C Syntax

```
int gssend_ (count, size_bytes, buffer)
```

```
int *count;  
int *size_bytes;  
int *buffer;
```

FORTRAN Syntax

```
INTEGER gssend (count, size_bytes, buffer)  
INTEGER count, size_bytes, buffer
```

Pascal Syntax

```
int_array = array [1..k] of integer;
```

```
FUNCTION gssend_ (
```

```
  VAR count : integer;  
  VAR size_bytes : int_array  
  VAR buffer : int_array ) : integer [PUBLIC];
```

Description

The **gssend** subroutine copies the number of bytes specified from the buffer into the adapter command interface queue. If the adapter does not have a queued command interface, this routine returns an unsuccessful return code.

As some display adapters require their queue interface to be loaded with a specific sized data such as halfword or fullword, the GSL will check the *size_bytes* parameter. If an integer number of such data units is not specified, the **GS-BUFS** return code is returned.

Passing display adapter level commands to the hardware requires detailed knowledge of the hardware and should be used with caution. Any adapter attributes set through the **gssend**

gssend

subroutine will not be restored by the GSL when a virtual terminal switch occurs. The contents of the buffer is not examined by the GSL for correctness, and therefore, no error checking, error detection, or error recovery is performed.

Before a FORTRAN program calls **gssend**, it must properly initialize both an array which points to the start of each command being passed to the display adapter hardware, and the commands themselves. The following sample program demonstrates how to initialize arrays and send two commands to a 5081 display adapter.

c The FORTRAN program stest loads 2 commands into the 5081 via gssend_().

```

        program stest
        integer count, size, savres, fildes
c declare the array that contains the number of bytes in each command
        integer numray(2)
c declare the array that contains the addresses of the commands
        integer bufaddr(2)
c declare the arrays of commands
        integer bufray(3,2)
        external gsinit, gsxptr, gscrls, gssend, gsterm

c define the number of commands to be sent to the 5081
        count = 2
c define the size of each command
        numray(1) = 10
        numray(2) = 10
c initialize the commands; each command displays a circle on the 5081
c command 1
c
        0x000a << 16 + 0x5006
        bufray(1,1) = 655360 + 20486
c
        512 << 16 + 512
        bufray(2,1) = 3554432 + 512
c
        50 << 16
        bufray(3,1) = 3276800
```



```
c command 2
c          0x00a << 16 + 0x5006
          bufray(1,2) = 655360 + 20486
c          512 << 16 + 512
          bufray(2,2) = 33554432 + 512
c 80 << 16
          bufray(3,2) = 5242880

c perform GSL initialization
          size = 0
          savres = -1
          fildes = -1
          call gsinit (buffer,size,savres,0,0,fildes)

c put the addresses of the commands in the buffer address array
          call gsxptr ( bufray(1,1), bufaddr(1) )
          call gsxptr ( bufray(1,2), bufaddr(2) )

c clear the screen
          rc = gsclr()

c send the commands to the 5081
          rc = gssend(count,numray,bufaddr)

c terminate GSL and exit
          rc = gsterm()

          stop
          end
```

Parameters

<i>count</i>	Specifies the number of elements in the <i>size-bytes</i> and <i>buffer</i> arrays. The count parameter should be greater than or equal to one.
<i>size-bytes</i>	Specifies the number of bytes printed by the corresponding element in the <i>buffer</i> array.
<i>buffer</i>	Each element of the buffer array points to a block of memory containing a command.

gssend

Return Value

GS_SUCC	Successful.
GS_BUFS	Invalid buffer size.
GS_USUC	Unsuccessful.

gstat

Purpose

Sets the text attributes for annotated text.

C Syntax

```
int gstat_ (color, page, baseline, font, name)
```

```
int *color, *page, *baseline, *font;  
char *name;
```

FORTRAN Syntax

```
INTEGER function gstat (color, page, baseline, font, name)
```

```
INTEGER color, page, baseline, font  
CHARACTER*n name
```

Pascal Syntax

```
FUNCTION gstat_ (
```

```
VAR color, page, baseline, font: INTEGER;  
VAR name: ARRAY [0..k] of CHAR  
): INTEGER [PUBLIC];
```

Description

The **gstat** subroutine defines the attributes for the class of text drawing functions.

Parameters

<i>color</i>	Specifies a text color entry in the color map. If it is -1, the attribute is unchanged.
<i>page</i>	Specifies the code page of a font for the display to use. The valid values for IBM supplied fonts are 0, 1, and 2 for code pages P0, P1, and P2, respectively. The value -1 indicates no change.

For printers and plotters, the *page* parameter is a font value specification. The IBM 3812 Pageprinter supports 128 different code pages. Again, the value -1 indicates no change.

baseline

Determines the direction of the text drawing. The valid values are:

- 1 Attribute remains unchanged.
- 0 Specifies 0 degrees, or left to right in the viewer's terms.
- 1 Specifies 90 degrees, or up in the viewer's terms.
- 2 Specifies 180 degrees, or right to left in the viewer's terms.

Note: The characters appear upside down.

- 3 For 270 degrees, or down in the viewer's terms.

If the baseline is other than 0 degrees and the font index is 0, then the font named by the *name* parameter must be a prerotated font. When a baseline change is made, another font path name is required.

font

Specifies, for displays, the font to use for text output operations. For printers and plotters, the *font* parameter specifies the vertical height of the font in pixels.

If the *font* index is -1, no change is made. If the *font* index is 0, then **gstat** uses the font specified by the *name* parameter. If the *font* index is a value from 1 to 14, the GSL uses one of the following predefined fonts:

Font Index	Width x Height (in pixels)	Style	Filename
1	9 x 20	Normal	nrm1.9x20
2	9 x 20	Italic	itl1.9x20
3	9 x 20	Bold	bld1.9x20
4	8 x 14	Normal	nrm1.8x14
5	4 x 8	Normal	nrm1.4x8
6	18 x 40	Normal	nrm1.18x40
7	12 x 30	Normal	nrm1.12x30
8	9 x 20	Ergonomic	erg1.9x20
9	6 x 9	Normal	nrm1.6x9
10	6 x 11	Normal	nrm1.6x11
11	7 x 15	Normal	nrm1.7x15
12	7 x 22	Normal	nrm1.7x22
13	11 x 23	Normal	nrm1.11x23
14	11 x 23	Bold	bld1.11x23

All left to right fonts are stored in the */etc/vtm* directory. Matching rotated fonts are stored in the */usr/lpp/gsl/fonts* directory.

name Contains the null-terminated full path name of the file used when the font attribute is specified as user. See "fonts" on page 3-79 for the format of this file.

If a single-shift control is outstanding and **gstat** is called to change the code page or the font, then the single-shift control is ignored.

(See "Code Page Switching" on page 4-9 for details about single-shift controls.)

For Pascal, the application must declare the arrays passed as being fixed length and declare the routine as accepting arrays of that length. The *k* in the routine declaration must be a constant.

Return Value

GS_SUCC	Successful.
GS_COLI	Invalid color index.
GS_CPID	Invalid code page identifier.
GS_BASL	Invalid baseline direction.
GS_FNTI	Invalid font index.
GS_FNTN	Invalid file name.

Related Information

In this book: "fonts" on page 3-79 and "Code Page Switching" on page 4-9.

gsterm

gsterm

Purpose

Terminates use of the GSL.

C Syntax

`void gsterm_ ()`

FORTRAN Syntax

`subroutine gsterm ()`

Pascal Syntax

`PROCEDURE gsterm_ () [PUBLIC];`

Description

The **gsterm** subroutine terminates the GSL. It deallocates any private storage required, returns the virtual terminal to KSR mode, and causes the monitor mode signals to be ignored.

gstext

Purpose

Writes annotated text.

C Syntax

```
int gstext_ (x, y, number, text)
```

```
int *x, *y, *number;  
char *text;
```

FORTRAN Syntax

```
INTEGER function gstext (x, y, number, text)
```

```
INTEGER x, y, number  
CHARACTER*n text
```

Pascal Syntax

```
FUNCTION gstext_ (
```

```
VAR x, y, number: INTEGER;  
VAR text: ARRAY [1..k] of CHAR  
): INTEGER [PUBLIC];
```

Description

The **gstext** subroutine writes the number of characters indicated by the parameters, starting at the specified baseline position and according to the relevant attributes. This subroutine is to be used only with annotated text.

The relevant attributes are:

- Color map
- Plane mask
- Font

- Code page
- Baseline direction
- Text color index.

Parameters

<i>x, y</i>	Define the baseline position for writing the text.
<i>number</i>	Indicates the number of bytes to write from the <i>text</i> string.
<i>text</i>	Contains the ASCII codes for the characters to write, as an array.

The graphics written to the frame buffer are determined by the 8-bit ASCII codes in the input data and the code page attribute. The ASCII control codes in between are ignored except the following: 1F, 1E, 1D, and 1C (hexadecimal). These control codes cause a shift to a predefined code page for the next ASCII character only. The code page definitions are:

1F	Bottom half of code page 1
1E	Top half of code page 1
1D	Bottom half of code page 2
1C	Top half of code page 2.

For any ASCII value between 0 and 31 (decimal), no graphic is written. For any other ASCII value and code page combination that does not result in a valid graphic, a dash is written.

For Pascal, the application must declare the array passed as being fixed length and declare the routine as accepting an array of that length; the *k* in the routine declaration must be a constant.

Return Value

GS-SUCC	Successful.
GS-CORD	Invalid coordinate.
GS-FBUF	Frame buffer overflow.
GS-INAC	Virtual terminal inactive.

gsulns

Purpose

Sets the user line pattern.

C Syntax

```
int gsulns_ (pattern, length, begin)
```

```
int *pattern, *length, *begin;
```

FORTRAN Syntax

```
INTEGER function gsulns (pattern, length, begin)
```

```
INTEGER pattern, length, begin
```

Pascal Syntax

```
FUNCTION gsulns_ (
```

```
  VAR pattern, length, begin: INTEGER  
): INTEGER [PUBLIC];
```

Description

The **gsulns** subroutine establishes the user line style.

Parameters

pattern Defines the pixel pattern used for the line style. A 1 bit indicates that the GSL draws a pixel; a 0 bit means that it does not.

length Defines the number of bits (starting with the most significant) of *pattern* used for line drawing. The bits are repeated for the length of the line.

The *length* parameter is a value not less than 2 or greater than 32. When using the 5081 display, the value of *length* must be 16.

gsulns

begin Indicates the length of the starting run of bits set to 1 in the pattern. It is used to adjust the beginning and ending runs of the noncontinuous line styles. The 5081 display does not use the *begin* parameter.

For proper appearance, the application-supplied line pattern should begin with a run of bits set to 1 and end with a run of bits set to 0.

Return Value

GS_SUCC	Successful.
GS LENG	Invalid length.

gsunlk

Purpose

Resumes signal processing.

C Syntax

`void gsunlk_ ()`

FORTRAN Syntax

`subroutine gsunlk ()`

Pascal Syntax

`PROCEDURE gsunlk_ () [PUBLIC];`

Description

The **gsunlk** subroutine indicates to the GSL that the application is finished with the display adapter and it can now read the **SIGRETRACT** signal.

The application supplied routine called at **SIGRETRACT** can be entered as a result of **gsunlk**.

gsvgrn

gsvgrn

Purpose

Sets the valuator granularity.

C Syntax

int gsvgrn- (*valuators*, *granularity*)

char **valuators*, **granularity*;

FORTRAN Syntax

INTEGER function gsvgrn (*valuators*, *granularity*)

CHARACTER *valuators*, *granularity*

Pascal Syntax

FUNCTION gsvgrn- (

VAR *valuators*, *granularity*: **CHAR**
); **INTEGER** [**PUBLIC**];

Description

The **gsvgrn** subroutine sets the resolution of input events generated by the valuator, that is, the number of events per turn of the valuator dial.

Parameters

- | | |
|--------------------|---|
| <i>valuators</i> | Specifies which valuator to set to the indicated granularity. Each bit in <i>valuators</i> corresponds to one of the valuator dials, with the most significant bit indicating that valuator 0 is to be set, the next most significant bit indicating that valuator 1 is to be set, and so on. |
| <i>granularity</i> | Specifies the desired resolution for the valuator indicated. It must have a value of 2 through 8 and indicates a resolution of 4, 8, 16, 32, 64, 128, or 256 points per revolution, respectively. The default value is 4, for a resolution of 16. |

An attempt to set the valuator granularity can fail for a variety of reasons, the most likely of which is that the device is not attached. The problem can be determined with a specific **ioctl** to the virtual terminal.

(See “hft” on page 5-39 for more information.)

Return Value

GS_SUCC	Successful.
GS_USUC	Unsuccessful.
GS_VALG	Invalid granularity.

Related Information

In this book: “hft” on page 5-39

gsxblt

gsxblt

Purpose

Moves a rectangular block in system or display adapter memory from one location to another.

C Syntax

```
int gsxblt_ (srcpix, dstpix, mskpix, W, H, logop)
```

```
int *srcpix, *dstpix, *mskpix, *W, *H, *logop;
```

FORTRAN Syntax

```
INTEGER function gsxblt (srcpix, dstpix mskpix, W, H, logop)
```

```
INTEGER srcpix (*), dstpix (*), mskpix (*) W, H, logop
```

Pascal Syntax

```
FUNCTION gsxblt_ (
```

```
VAR srcpix, dstpix, mskpix: ARRAY [32] of INTEGER;
```

```
VAR W, H, logop : INTEGER
```

```
): INTEGER [PUBLIC];
```

Description

The **gsxblt** subroutine moves a rectangular block of pixels from one memory location to another, either in system memory or in the display adapter frame buffer.

For FORTRAN specific address information, see “gsxptr” on page 6-173.

The **gsxblt** subroutine is used to support windowing operations, such as overlays and movement around the screen. The source rectangle and the destination rectangle can be in either system or adapter pixel memory. The **gsxblt** subroutine is also used for user defined cursors and the save and restore of a pixel map for applications like pop-up menus.

The mask operation provided by the **gsxblt** subroutine controls which pixels in the destination rectangle can be modified.

The relevant attributes are:

- Plane mask
- Color map.

Parameters

srcpix Contains the address of the source pixel map.

dstpix Contains the address of the destination pixel map.

mskpix Contains the address of the mask operation pixel map. This parameter should equal 0 if there is no bit mask operator to apply. For FORTRAN applications, a valid *mskpix* array must always be defined. If no masking is required, the address field of the array, **mskpix[9]**, must be initialized to 0.

The *mskpix* pixel map must always consist of only 1 bit per pixel, and the mask rectangle must always be the same size as the source and destination rectangles. In the mask rectangle, a 1 bit means that the corresponding pixel in the destination rectangle can be modified, while a 0 bit means the destination pixel will not be modified.

W Defines the width of the rectangular area to be transferred.

H Defines the height of the rectangular area to be transferred.

logop Indicates the logical operation to perform between the source pixel map and the destination pixel map.

In the following table, please note:

- The source or tile (a special type of source) pixels represent bits of data to be merged in some way with the corresponding bits of data in the destination rectangle.
- The first three columns of the table specify the operations you can perform, and the **Code** column contains the corresponding value you should specify for the *logop* parameter.
- There are two unique codes for each logical operation, to be used depending on whether the tiling bit in the source pixel map is set. Codes 0-15 must be used when the tiling bit is not set, while codes 16-31 must be used when the tiling bit is set.
- A ~ (tilde) represents the logical INVERSE.

Type of Source	Logical Operation	Type of Destination	Code
		Destination clear	0
		Set Destination	15
	No operation	Destination	5
		~Destination	10
Source	REPLACE	Destination	3
Source	AND	Destination	1
Source	AND	~Destination	2
Source	Exclusive-OR	Destination	6
Source	OR	Destination	7
Source	OR	~Destination	11
~Source	REPLACE	Destination	12
~Source	AND	Destination	4
~Source	AND	~Destination	8
~Source	Exclusive-OR	Destination	9
~Source	OR	Destination	13
~Source	OR	~Destination	14
		Destination clear	16
		Set Destination	31
	No operation	Destination	21
		~Destination	26
Tile	REPLACE	Destination	19
Tile	AND	Destination	17
Tile	AND	~Destination	18
Tile	Exclusive-OR	Destination	22
Tile	OR	Destination	23
Tile	OR	~Destination	27
~Tile	REPLACE	Destination	28
~Tile	AND	Destination	20
~Tile	AND	~Destination	24
~Tile	Exclusive-OR	Destination	25
~Tile	OR	Destination	29
~Tile	OR	~Destination	30

A *pixel map* is a 32-bit array of integers that contains the following fields:

0 Device ID (0 for memory)

1 Flags

In the following explanations, bit 0 is the low-order bit.

- Plane (XY) format is selected when bit 0 is set and bits 1 and 2 are not set. Pixel (Z) format is selected when bits 0, 1, and 2 are not set.
- A repetitive tile is specified when bit 3 is set, while no tile is specified when bit 3 is not set.

If the repetitive tile bit is set in the *srcpix*, pixel map, then the **Device ID** field in that pixel map must equal 0. The tile data must be in memory.

- Bit 4 selects the lower-left coordinate system when it is set, and the upper-left coordinate system when it is not set.

2 Height (in pixels)

3 Width (in pixels)

This value must be an even multiple of 16 pixels for all pixel maps, which means that all pixel maps must be at least 16 pixels wide.

4 Number of bits per pixel

5 Pixels per byte, right justified

6 Bytes per pixel

7 x offset

8 y offset

9 Address of upper-left corner of data

10 Foreground color index

11 Background color index

12 - 31 Reserved.

Definitions of pixel map terms include:

Device ID

This is a required parameter for all pixel map definitions. If the pixel map being defined is a display adapter, this field must contain the Device ID of that display adapter. If the pixel map resides in system memory, then this field must equal 0.

Pixel format

Data stored in this format has all bits for a pixel stored together. The data starts with the origin and increases first in the *x* direction, then in the *y* direction.

As an example using the upper-left coordinate system, a pixel map with 4 bits per pixel and 1 pixel per byte stores the 4 bits for the pixel at location (0,0) in the first byte of the data area, right justified in the byte. The 4 bits for the pixel at location (1,0) are stored in the second byte, followed by the rest of the pixel values in that row. When the end of the row is reached, the next byte contains the 4 bits for the pixel at location (0,1), followed by the rest of the pixel values in that row, and so on for the entire image.

Plane format

Plane format indicates that each of the bits that make up a pixel is stored in a separate, consecutive plane in memory. The most significant bit is first, followed by the next significant, and so on to the least significant bit, which is last. The bits within a plane are packed together 8 bits per byte. Therefore, using the upper-left coordinate system as an example, a pixel map with 4 bits per pixel would consist of four separate planes of data with the first bit value being the one for location (0,0) and increasing first in the *x* direction, then in the *y* direction.

Repetitive tiling operation

This operation consists of repeatedly copying a 16-pixel wide by 16-pixel high tile rectangle pointed to by the tile pixel map data address to fill a rectangular area of a size specified by the **H** and **W** parameters of this call. The format of the tile data is determined by the format defined in the **flags** field of the tile pixel map structure.

Upper-left coordinate system

This indicates that the upper-left corner of the pixel map is used as the origin of the coordinate system, with increasing values of *x* moving to the right and increasing values of *y* moving down. The *x* offset and *y* offset are to set the upper-left corner of the rectangle when using this coordinate system.

Lower-left coordinate system

This indicates that the lower-left corner of the pixel map is used as the origin of the coordinate system, with increasing values of *x* moving to the right and increasing values of *y* moving up. The *x* offset and the *y* offset are set to the lower-left corner of the rectangle when using this coordinate system. Note, however, that the data address specified in the pixel map structure must always point to the upper-left corner of the data area no matter which coordinate system is defined.

Number of bits per pixel

This field identifies the number of bits of data required to define a pixel value. For example, a simple monochrome display requires only 1 bit per pixel, while a color display may require 4 bits of information to define a pixel.

Number of pixels per byte

If the number of bits per pixel is less than 8, this field defines how many pixels are stored in each byte of pixel map data. A pixel map with only 1 bit per pixel must always store 8 pixels per byte. It is strongly recommended that for between 2 and 7 bits per pixel, you store data with only 1 pixel per byte.

Bytes per pixel

If the number of bits per pixel is greater than 8, this field defines how many bytes are used to store each pixel. It is strongly recommended that for between 9 and 16 bits per pixel, you store data 2 bytes per pixel. For between 17 and 32 bits per pixel, data should be stored 4 bytes per pixel.

Foreground color index

This specifies the color index value to use for a value of 1 in the source pixel map during a color expansion operation.

Background color index

This specifies the color index value to use for a value of 0 in the source pixel map during a color expansion operation.

A **color expansion operation** takes place automatically when the source pixel map data area contains only 1 bit per pixel and the destination pixel map data area is a color display adapter frame buffer defined to have more than 1 bit per pixel. In this case, when a 1 is specified in the source pixel map data area, the foreground color index value specified in the destination pixel map (dstpix) is written to the destination data area. When a 0 is specified in the source pixel map data area, the background color index value specified in the destination pixel map (dstpix) is written to the destination data area.

The foreground color index and the background color index must be initialized in the dstpix pixel map before calling this operation, but do not need to be initialized in the srcpix or mskpix pixel maps.

Not all logical operations are supported for a color expansion operation. The following table shows which operations are supported. In this table, a ~ (tilde) represents the logical INVERSE. Note that the operations in the left column of the table are for source pixel maps, while the operations in the right column are for tile pixel maps.

Type of Operation	Code	Type of Operation	Code
Destination clear	0	Destination clear	16
Set destination	15	Set destination	31
Destination	5	Destination	21
~Destination	10	~Destination	26
Source	3	Tile	19
~Source	12	~Tile	28

If a source or destination pixel map structure defines the active display adapter, you do not need to initialize all the fields of that pixel map structure. Device-dependent information, such as height, width, pixels per byte, bytes per pixel, and address of data, is supplied automatically. You must initialize the fields for device ID, bits per pixel, flags (except for the data format bits), x offset, and y offset. Also, the foreground color index and the background color index must be initialized if appropriate for this adapter.

When initializing a pixel map structure to use as the mskpix parameter:

gsxbt

1. The flags field should equal a value of 0x01 if the upper-left coordinate system will be used or 0x11 if the lower left coordinate system will be used.
2. The number of bits per pixel must equal 1.
3. The number of pixels per byte must equal 8.

The GSL plane mask attribute applies to all **gsxbt** operations that use the display adapter as the source or destination pixel map.

The GSL color map attribute applies to all **gsxbt** operations that use the display adapter as the destination pixel map.

Return Value

GS-SUCC	Successful.
GS-IWID	Invalid width specification. The x -offset plus the W parameter of one of the pixel maps exceeds the total width of that pixel map.
GS-IHEI	Invalid height specification. The y -offset plus the H parameter of one of the pixel maps exceeds the total height of that pixel map.
GS-NPLF	Source and destination data formats do not match.
GS-INAC	Virtual terminal inactive.
GS-CORD	Invalid coordinate specified that placed the origin of the source, destination, or mask rectangle outside its pixel map.
GS-IBPP	Invalid value for bits per pixel in the source pixel map.
GS-CEXP	Color expansion operation attempted, but the destination pixel map was not a display adapter.
GS-PWID	The width of one of the pixel maps is not an even multiple of 16 pixels.

Related Information

In this book: “gsxptr” on page 6-173.

gsxcnv

Purpose

Converts pixel map data organization.

C Syntax

```
int gsxcnv_ (inppix, outpix)
```

```
int *inppix, *outpix;
```

FORTRAN Syntax

```
INTEGER function gsxcnv (inppix, outpix)
```

```
INTEGER inppix (*), outpix (*)
```

Pascal Syntax

```
FUNCTION gsxcnv_ (
```

```
  VAR inppix, outpix: INTEGER  
): INTEGER [PUBLIC];
```

Description

The **gsxcnv** subroutine converts pixel map data to and from planes. That is, **gsxcnv** converts XY form to and from pixels, or Z form.

See "gsxblt" on page 6-164 for more information on data formats and the definition of a pixel map.

Parameters

<i>inppix</i>	Points to the address of the pixel map whose data area is to be converted.
<i>outpix</i>	Points to the address of the pixel map that contains the address of where to put the converted data.

Both the *inppix* and *outpix* parameters contain the address of a pixel map. The fields of each pixel map must be completely initialized before calling this subroutine. Both pixel maps must point to data areas that reside in system memory, not in a display adapter frame buffer.

The *inppix* and *outpix* pixel maps do not have to specify the same number of bits per pixel. If there are more input bits per pixel, the least significant bits are truncated. If there are less input bits per pixel than required to fill out the destination, the most significant bits are filled with zeros.

The **gsxcnv** subroutine only supports pixel maps defined to have 8 bits per pixel or less. If a pixel format pixel map is defined with less than 8 bits per pixel, the data must be arranged 1 byte per pixel, right justified in that byte.

The widths and heights of the two data areas must be identical.

Warning: The calling process must allocate enough storage in the area pointed to by the *outpix* pixel map to contain all of the converted data.

Return Value

GS_SUCC	Successful.
GS_INPF	Invalid data format specified in <i>inppix</i> pixel map structure.
GS_OUTF	Invalid data format specified in <i>outpix</i> pixel map structure.
GS_BMAX	Pixel map defines data of more than 8 bits per pixel.

Related Information

In this book: “gsxblt” on page 6-164.

gsxptr

Purpose

Handles FORTRAN addressing of data.

C Syntax

None

FORTRAN Syntax

INTEGER function **gsxptr** (*intptr*, *datptr*)

INTEGER *intptr* (*), *datptr* (*)

Pascal Syntax

None

Description

The **gsxptr** subroutine places a data address in a variable so that the data address field of a pixel map structure can be initialized.

In a FORTRAN application, you must first call the **gsxptr** subroutine, then the **gsxblt** subroutine.

For C and Pascal applications and for more information, see “gsxblt” on page 6-164.

Parameters

intptr Contains the address of the variable containing the data area.

datptr Will be initialized to the address of the data area.

gsxptr

Return Value

GS-SUCC Successful.

Related Information

In this book: “gsxbt” on page 6-164.

gsxtat

Purpose

Sets the text attributes for annotated text using the rfont format.

C Syntax

`int gsxtat_ (foreground, background, logop, font, clipbox)`

`int *foreground, *background, *logop, *font, *clipbox;`

FORTRAN Syntax

`INTEGER function gsxtat (foreground, background, logop, font, clipbox)`

`INTEGER foreground, background, logop, font, clipbox`

Pascal Syntax

`FUNCTION gsxtat_ (`

`VAR foreground, background, logop, : INTEGER;`

`VAR font: ARRAY [1..k] of INTEGER;`

`VAR clipbox: ARRAY [1..l] of INTEGER;`

`): INTEGER [PUBLIC];`

Description

The **gsxtat** subroutine defines the attributes to be used when drawing text with a font in the rfont format.

Parameters

foreground Defines an entry in the color map to use for the foreground color (bits set to 1) in the font raster for each character. A value of -1 indicates no change for this attribute.

background Defines an entry in the color map to use for the background color (bits set to 0) in the font raster for each character. A value of -1 indicates no change for this attribute.

logop

Indicates the logical operation to perform between the font raster and the display destination.

In the following table, please note:

- The source pixels represent bits of data from the font raster to be merged in some way with the corresponding bits of data in the destination rectangle.
- The first three columns of the table specify the operations you can perform, and the **Code** column contains the corresponding value you should specify for the *logop* parameter.
- A ~ (tilde) represents the logical INVERSE.

Type of Source	Logical Operation	Type of Destination	Code
		Destination clear	0
		Set Destination	15
	No operation	Destination	5
		~Destination	10
Source	REPLACE	Destination	3
Source	AND	Destination	1
Source	AND	~Destination	2
Source	Exclusive-OR	Destination	6
Source	OR	Destination	7
Source	OR	~Destination	11
~Source	REPLACE	Destination	12
~Source	AND	Destination	4
~Source	AND	~Destination	8
~Source	Exclusive-or	Destination	9
~Source	OR	Destination	13
~Source	OR	~Destination	14

A value of -1 for this parameter indicates no change in the current logical operation.

font

Points to a data area that contains the font header and raster definitions for all characters defined in the font. The calling process is responsible for either mapping the font file or copying it into a memory area in order to obtain a pointer to the data area.

Setting the value of this pointer to 0 indicates no change to the current font file.

The GSL supports only a subset of the different forms that the rtfon format allows. Specifically, the GSL supports any combination of the following font formats:

- Fixed width and height

- Variable width and/or height
- Halfword alignment or fullword alignment
- Glyphs in raster format only
- Index character array width of 4 bytes
- All individual glyph character bounds for variable width and height fonts, except negative left or right bearings.

For more information on valid formats for rtfon files, see “fonts” on page 3-79 and the **rtfont.h** header file.

The GSL does not support any formats for rtfon files other than those listed above. If the font file specified is not in a supported format, then the GSL returns the **GS_FFMT** return code.

clipbox

Specifies an array of integers that correspond to a rectangular area on the display screen. When the **gsxtxt** subroutine is used to draw text, any full or partial characters that fall outside this area are clipped. The elements of the area to clip are as follows:

First element	Reserved. This value should always be 1.
Second element	Specifies the x coordinate of the lower left corner of the clip box, in pixels.
Third element	Specifies the y coordinate of the lower left corner of the clip box, in pixels.
Fourth element	Specifies the height of the clip box, in pixels.
Fifth element	Specifies the width of the clip box, in pixels.

The bottom and left edges of the clip box are inclusive, while the top and right edges are exclusive.

This parameter is a pointer to the clip box array, which is not copied into any GSL data structure, allowing the calling process to modify the elements of the array without calling the **gsxtat** subroutine. If the values for the clip box are changed between calls to the **gsxtxt** subroutine, the new clip box is used for all text drawing until another change is made.

Setting the value of this pointer to 0 indicates no change.

Warning: Since the GSL subroutines that use the rtfon format are designed for high-performance text drawing, no verification is made of the validity of the clip box. It is the responsibility of the calling process to ensure that the entire clip box resides inside the physical size of the display. Using a clip box that is not entirely within the screen will produce unpredictable results.

gsxtat

When the GSL is installed from diskette, an attempt is made to convert the 14 supplied VRM format fonts into rtfon format. The **vrn2rtfon** command (described in the *AIX Operating System Commands Reference*) is used on the 14 VRM fonts in the **/etc/vtm** directory, and the resulting rtfons are stored in the **/usr/lpp/fonts** directory. The following list shows the rtfon format files stored in **/usr/lpp/fonts**:

Width x Height (in pixels)	Style	File name
4 x 8	Normal	Rom6.500
6 x 9	Normal	Rom7.500
6 x 11	Normal	Rom8.500
7 x 15	Normal	Rom11.500
7 x 22	Normal	Rom16.500
8 x 14	Normal	Rom10.500
9 x 20	Normal	Rom14.500
9 x 20	Italic	Itl14.500
9 x 20	Bold	Bld14.500
9 x 20	Ergonomic	Erg14.500
11 x 23	Normal	Rom17.500
11 x 23	Bold	Bld17.500
12 x 30	Normal	Rom22.500
18 x 40	Normal	Rom29.500

All of these fonts have fixed width and height and are halfword aligned.

For Pascal, the application must declare the arrays passed as being fixed length and declare the routine as accepting arrays of that length. The *k* and *l* in the routine declaration must be constants.

Return Value

GS_SUCC	Successful.
GS_FFMT	Invalid font format.
GS_LONS	Invalid logical operation.

File

/usr/include/rtfont.h

Related Information

In this book: “fonts” on page 3-79 and “gsxtxt” on page 6-180.

The **vrn2rtfont** command in *AIX Operating System Commands Reference*.

gsxtxt

gsxtxt

Purpose

Writes annotated text using the rtfont format.

C Syntax

```
int gsxtxt_ (x, y, number, text)
```

```
int *x, *y, *number;  
char *text;
```

FORTRAN Syntax

```
INTEGER function gsxtxt (x, y, number, text)
```

```
INTEGER x, y, number  
CHARACTER*n text
```

Pascal Syntax

```
FUNCTION gsxtxt_ (
```

```
VAR x, y, number: INTEGER;  
VAR text: ARRAY [1..k] of CHAR  
): INTEGER [PUBLIC];
```

Description

The **gsxtxt** subroutine displays the specified text string using the rtfont format. Only those full or partial characters that fall within the clip box specified by the **gsxtat** subroutine are displayed. In addition, since there is no default rtfont defined for use by the GSL, the **gsxtat** subroutine must be called to set all relevant attributes before the first call to this subroutine.

(See "gsxtat" on page 6-175 for additional information.)

The relevant attributes are:

- xtext foreground color
- xtext background color
- xtext logical operation

- xtext clip box
- xtext current rtfont file
- Plane mask
- Color map.

Parameters

<i>x, y</i>	Define the baseline position for writing the text.
<i>number</i>	Indicates the number of bytes to write from the <i>text</i> string.
<i>text</i>	Contains the ASCII codes for the characters to write, as an array.

For Pascal, the application must declare the array passed as being fixed length and declare the routine as accepting an array of that length; the *k* in the routine declaration must be a constant.

Return Value

GS_SUCC Successful.

Since the text is either displayed or clipped in any case, the **gsxtxt** subroutine always completes with a successful return code.

File

/usr/include/rtfont.h

Related Information

In this book: “fonts” on page 3-79 and “gsxtat” on page 6-175.

Appendix A. Error Codes

This section describes the error conditions that can occur when using the system calls described in this book. Some subroutines also use these codes to indicate errors.

System calls indicate the fact that an error has occurred by returning a special value. This value is almost always -1, but check the description of the individual system call to be sure. Also, a number that identifies the error is stored in an external variable named **errno**. The **errno** variable is not cleared when a system call finishes successfully, so its value is meaningful only after an error has occurred.

If you are going to check the value of **errno** in a program, include the following line at the top of the source file:

```
#include <errno.h>
```

The **errno.h** header file declares the **errno** variable and defines the name of each error condition.

For each error code, the following list shows the symbolic name defined in the **/usr/include/errno.h** header file, the corresponding numeric value, and a brief description of the error:

EPERM (1)

Not owner

Cause: You attempted to modify a file in some way forbidden except to the owner of the file or to superuser. Or, a user other than superuser attempted to do something that only superuser is allowed to do.

ENOENT (2)

No such file or directory

Cause: The file specified does not exist, or one of the directories in a path name does not exist.

ESRCH (3)

No such process

Cause: A process, corresponding to that specified in the *pid* parameter of the **kill** or **ptrace** system calls, cannot be found.

EINTR (4)**Interrupted system call**

Cause: An asynchronous signal (such as interrupt or quit), which you have elected to catch, occurred during a system call. If the system call resumes after processing the signal, it appears as if the interrupted system call returned this error condition.

EIO (5)**I/O error**

Cause: A physical I/O error occurred. In some cases, this error occurs on a system call following the one to which it actually applies.

ENXIO (6)**No such device or address**

Cause: I/O on a special file referred to a device or subdevice that does not exist or referred to an address that is beyond the limits of the device.

E2BIG (7)**Arg list too long**

Cause: The combined length of the argument list and the environment list passed to one of the **exec** system calls totaled more than 5,120 bytes.

ENOEXEC (8)**Exec format error**

Cause: A request was made to execute a file that has the appropriate permissions, but does not start with either a valid shared library magic number, or a valid text header. (For information about text headers, see "a.out" on page 3-5.)

EBADF (9)**Bad file number**

Cause: A file descriptor was specified that does not refer to an open file, or a read request was made to a file that is open only for writing, or a write request was made to a file that is open only for reading.

ECHILD (10)**No child processes**

Cause: A process that invoked the **wait** system call has no existing child processes that have not been waited for.

EAGAIN (11)**No more processes**

Cause: The **fork** system call failed because the system's process table is full or the user is not allowed to create any more processes. Or, an attempt was made to access a region of a file that has an outstanding enforcement-mode lock. (See "lockf" on page 2-427 about file locking.)

ENOMEM (12)**Not enough space**

Cause: During a **brk**, **sbrk**, or **exec** system call, a program asked for more space than the system is able to supply. This is not a temporary condition. The maximum space size is a system parameter.

EACCES (13)**Permission denied**

Cause: An attempt was made to access a file in a way that is forbidden by the protection system.

EFAULT (14)**Bad address**

Cause: An address passed to a system call that points to a location outside of the process's allocated address space.

ENOTBLK (15)**Block device required**

Cause: A nonblock file was specified when a block device is required, such as in the **mount** system call.

EBUSY (16)**Device busy**

Cause: An attempt was made to mount a device that is already mounted, or an attempt was made to dismount a device on which there is an active file. This error also occurs when an attempt is made to enable accounting when it has already been enabled.

EEXIST (17)**File exists**

Cause: An existing file was specified to a system call or subroutine that would create that file, such as the **link** system call.

EXDEV (18)**Cross-device link**

Cause: An attempt was made to link to a file on another device. (See “link” on page 2-417.)

ENODEV (19)**No such device**

Cause: An attempt was made to use an inappropriate system call to a device, for example, to write to a read-only device.

ENOTDIR (20)**Not a directory**

Cause: A nondirectory parameter was specified where a directory is required, for example in a path prefix or as a parameter to the **chdir** system call.

EISDIR (21)**Is a directory**

Cause: An attempt was made to write on a directory.

EINVAL (22)**Invalid argument**

Cause: An invalid argument or action was specified to a system call, such as dismounting a device that is not mounted, specifying an undefined signal, or writing to a file for which **lseek** has generated a negative file pointer.

ENFILE (23)**File table overflow**

Cause: An attempt was made to open a file, and the system's table of open files is full.

EMFILE (24)**Too many open files**

Cause: A process attempted to open more than two hundred (200) file descriptors at one time.

ENOTTY (25)**Not a typewriter**

Cause: An `ioctl` system call was issued to a special file that does not support `ioctl`.

ETXTBSY (26)**Text file busy**

Cause: This error occurs when an attempt is made to execute a pure-procedure program or shared library that is currently open for writing or reading. It also occurs when an attempt is made to open for writing or to remove a pure-procedure program or shared library while that program or library is being executed.

EFBIG (27)**File too large**

Cause: The size of a file exceeded the maximum file size (1,082,201,088 bytes), or the maximum size set by the `ulimit` system call.

ENOSPC (28)**No space left on device**

Cause: During a write to an ordinary file, the device ran out of free space on the file system where the file resides.

ESPIPE (29)**Illegal seek**

Cause: An `lseek` system call was issued to an unseekable file or device, such as a pipe.

EROFS (30)**Read-only file system**

Cause: An attempt was made to modify a file or directory on a device that is mounted as read-only.

EMLINK (31)**Too many links**

Cause: An attempt was made to make more than the maximum number of links (1000) to a file.

EPIPE (32)**Broken pipe**

Cause: An attempt was made to write to a pipe for which there is not a process to read the data. This condition normally generates a signal; the error is returned if the signal is ignored.

EDOM (33)**Argument out of domain**

Cause: A parameter to a Math Library (**libm.a**) subroutine was out of the domain of the function.

ERANGE (34)**Result too large**

Cause: The return value of a Math Library (**libm.a**) subroutine is not representable within machine precision.

ENMSG (35)**No message of desired type**

Cause: An attempt was made to receive a message of a type that does not exist on the specified message queue.

EIDRM (36)**Identifier removed**

Cause: The specified identifier has been removed from the file system's name space. (See "msgctl" on page 2-463, "semctl" on page 2-707, and "shmctl" on page 2-738.)

Note: The values **ECHRNG (37)** through **EL2HLT (44)** are supplied in the **errno.h** header file for compatibility with UNIX System V. These values are not set by any AIX software.

ECHRNG (37)	Channel number out of range
EL2NSYNC (38)	Level 2 not synchronized
EL3HLT (39)	Level 3 halted
EL3RST (40)	Level 3 reset
ELNRNG (41)	Link number out of range
EUNATCH (42)	Protocol driver not attached
ENOC SI (43)	No CSI structure available
EL2HLT (44)	Level 2 halted

EDEADLK (45)**Deadlock condition if locked**

Cause: A potential deadlock was detected while attempting to lock a region of a file with the **lockf** system call.

ENOTREADY (46)**Device not ready**

Cause: The device is not ready for operation. For example, a diskette drive does not contain a diskette, or the device is not powered on.

EWRPROTECT (47)**Write-protected media**

Cause: The I/O media is write-protected.

EFORMAT (48)**Unformatted or incompatible media**

Cause: The I/O media has not been formatted or the format is not compatible with the I/O device.

ENOLCK (49)**No locks available**

Cause: There are no more file locks available. Too many segments are already locked.

ENOCNNECT (50)**Cannot establish connection**

Cause: A new network connection to a remote node cannot be made.

EBADCONNECT (51)**Connection down**

Cause: An attempt to use an existing connection to a remote node has failed.

ESTALE (52)**Missing file or filesystem**

Cause: The file system of a remote file has been unmounted, or the file descriptor of a remote file has become obsolete.

EDIST (53)**Requests blocked by Administrator**

Cause: Sending or receiving of requests is currently not allowed. All requests may be blocked or only requests of the specified type.

EWOULDLOCK (54)**Operation would block**

Cause: The socket is not blocking because **O_NDELAY** is set, but the desired kind of data is not available or, for an **accept** operation, no connections are pending.

EINPROGRESS (55)**Operation now in progress**

Cause: The socket was marked **O_NDELAY** by an **fcntl** system call, then a **connect** operation was attempted that has not completed yet.

EALREADY (56)**Operation already in progress**

Cause: The requested socket connection or disconnection is already in progress.

ENOTSOCK (57)**Socket operation on non-socket**

Cause: The command cannot complete because the file descriptor specified is not a socket.

EDESTADDRREQ (58)**Destination address required**

Cause: The attempted socket operation failed because a destination address was required, but not provided.

EMSGSIZE (59)**Message too long**

Cause: The socket data transfer failed because the message exceeded the size limits.

EPROTOTYPE (60)**Protocol wrong type for socket**

Cause: Either the two sockets to be connected are not of the same type, or the protocol used does not support this type of socket.

ENOPROTOOPT (61)**Protocol not available**

Cause: The protocol specified either does not support this particular option or does not support any options.

EPROTONOSUPPORT (62) Protocol not supported

Cause: No protocol of the specified type and domain exists.

ESOCKTNOSUPPORT (63) Socket type not supported

Cause: The type of socket specified is not supported. Do not use this type of socket in your program.

EOPNOTSUPP (64) Operation not supported on socket

Cause: This socket, with its particular type, domain, and protocol, does not allow the requested operation.

EPFNOSUPPORT (65) Protocol family not supported

Cause: The socket protocol specified is not supported. Do not use this protocol in your program.

EAFNOSUPPORT (66) Address not supported by protocol family

Cause: The socket name is of a type that is not valid in this socket or the domain.

EADDRINUSE (67) Address already in use

Cause: A **bind** or **connect** operation was attempted using a socket name that is already in use.

EADDRNOTAVAIL (68) Can't assign requested address

Cause: The requested socket name is not available to this machine. Either an incorrect socket name was used, or there is a problem at the remote node where the socket name should be.

ENETDOWN (69)**Network is down**

Cause: A socket operation failed because the network is down.

ENETUNREACH (70)**Network is unreachable**

Cause: A socket operation failed because the destination is at a remote node that cannot be reached over the network.

ENETRESET (71)**Network dropped connection on reset**

Cause: The host the socket was connected to went down. The connection can be reestablished after the remote node is restarted.

ECONNABORTED (72)**Software caused connection abort**

Cause: The connection between a socket and a remote node was terminated at the local node, the remote node, or the network level.

ECONNRESET (73)**Connection reset by peer**

Cause: The connection with another socket was reset by that socket. This **errno** can be set due to an error, or just due to a connection that was closed.

ENOBUFS (74)**No buffer space available**

Cause: Not enough buffer space is available for the requested socket operation.

EISCONN (75)**Socket is already connected**

Cause: A **connect** operation was attempted on a socket that is already connected.

ENOTCONN (76)**Socket is not connected**

Cause: A socket operation other than a **connect** was attempted on a socket that is not currently connected, or a **send** operation that does not require a connection was attempted without a destination address.

ESHUTDOWN (77)**Can't send after socket shutdown**

Cause: An attempt was made to send data after a **shutdown** operation was done on the socket.

ETIMEDOUT (78)**Connection timed out**

Cause: A remote socket did not respond within the timeout period set by the protocol of the socket on this node.

ECONNREFUSED (79)**Connection refused**

Cause: A remote node refused to allow the attempted **connect** operation.

EHOSTDOWN (80)**Host is down**

Cause: A socket operation failed because the remote node specified is down.

EHOSTUNREACH (81)**No route to host**

Cause: A socket operation failed because no route to the remote node was available due to an incorrect address, an incorrect routing table, or network hardware problems.

ELOOP (85)**Unable to complete translation**

Cause: More symbolic links than allowed when translating path name.

ENAMETOOLONG (86) **Path name exceeds maximum number of characters**

Cause: A component of a path name exceeds 5 characters, or an entire path name exceeds 1023 characters.

ENOTEMPTY (87) **Directory not empty**

Cause: You attempted to delete a directory that was not empty with either **rmdir** or **rename**

EDQUOT (88) **Disc quota exceeded**

Cause: User's allocation of disc block on a remote file system has been exceeded.

EREMOTE (93) **Too many levels of remote in path.**

Cause: A path name crosses too many filesystems.

Appendix B. Writing a Queuing System Backend

This section assumes that you know what a queue backend is (friendly and unfriendly) and that you need to write one. This appendix discusses only friendly backends, not backends in general; so *backend* in this chapter refers to friendly backends. See *Managing the AIX Operating System* for more information about backends.

Introduction

The principal purpose of a backend is to send characters to a device, typically a printer. There are several ways the backend can do this. First, it can open a particular device and write to it. This has the advantage of simplicity, but it means that the backend cannot be used for any other device. Second, it can accept a parameter supplied by the user to tell it which device to use. This is more flexible, but involves a little extra work. Third, it can simply write to its standard output, and the **qdaemon** command will automatically open the device onto the correct file descriptor. This is the recommended method. It works only if the **file** field in the **qconfig** file has been set up appropriately.

The backend is invoked once for every file or group of files to be printed. The name of each file to be printed is passed to the backend as a parameter. The backend must open the file, read its contents, and send them to the device in one of the ways previously described.

Since the backend must open files, read them, and write to devices, you (the writer of a backend) should understand the domain where the backend operates. When a backend is invoked, its current directory is the one where the print request was made. The name of the file or files to be printed can either be a direct or relative path name. The UID and GID of the backend are those of the process that invoked the **print** command.

If the backend writes to its standard output and allows the **qdaemon** process to open the device, permissions are handled by the **qdaemon** automatically. Otherwise, the backend will need to have write permission on the special file corresponding to the device. This may require changing the protections on the device or installing the backend set-user-ID or set-group-ID.

By default, **stdin**, **stdout**, and **stderr** are all open to the null device (**/dev/null**), though it is possible to override the setting of **stdout** (and possibly **stdin**) with the **file** and **access** lines in the **qconfig** file.

Interaction Between Qdaemon and Backend

Besides reading files and writing to devices, a friendly backend must cooperate with the **qdaemon** in several ways. The requirements can be summarized as follows:

- Recognize a **-statusfile** parameter and call a library routine that does some initialization.
- Print burst pages as requested.
- Print extra copies as requested.
- Update status information (pages printed, percentage done) periodically.
- Supply charges (accounting data) for the completed job.
- Exit with some agreed on codes.
- Pass error messages through a special routine.
- Set state to WAITING, if appropriate.
- Terminate cleanly on receipt of SIGTERM.

Each requirement is discussed more fully in the following.

There is a set of library routines that the backend should use to fulfill these requirements. The routines were designed to make the task of writing a backend as easy as possible. These routines are in the **/lib/libqib.a** library and accessible with the **-lib** flag. The individual routines are discussed in the body of the text that follows, and a summary table is given at the end of this chapter.

The **-statusfile** Parameter

When the **qdaemon** process invokes a backend, it passes the following parameters, in order:

1. The parameters appearing in the **qconfig** file
2. The **-statusfile** parameter, if running as a friendly backend
3. The flags that the **print** command did not recognize, in the order they were given
4. The names of one or more files to be printed.

The presence of the **-statusfile** parameter indicates that the status file is open on file descriptor 3 of the backend.

The status file provides a means for the **qdaemon** process and the backend to communicate. The daemon passes such information as the date of the file, which burst pages are to be printed, the number of copies to be printed, and so on. The backend passes back the charge for the job it has just finished running. In addition, the backend periodically writes into the file the number of pages it has printed and what percent of the job is finished. This information is read by the **print -q** command.

Backends should never explicitly write into their status file. Instead, they should call the library routines that do so. The reason for calling the routines is twofold: (1) backends are spared the trouble of accessing the status file directly, and (2) the format of the status file can be changed without requiring backends to be rewritten. In this case, the backends only need to be re-linked.

To initialize certain data common to the library routines, the backend must call the routine **log_init**. The call is:

```
log_init();
```

This routine should be called when the **-statusfile** parameter is recognized. The **log_init** routine, like all the routines in library whose names begin **log-**, returns a value of -1 if it fails.

Burst Pages

There are four types of burst pages:

- header** A page preceding a file that shows its title, date, its recipient, and other information.
- trailer** A page following a file that gives the name of the user of the output.
- feed** Blank pages printed only when the printer has become idle. Feed pages make it easier for users to tear off paper from the printer.
- align** A form-feed printed only when the printer has been idle and is about to print a new job. The form-feed aligns the paper to top-of-form and is helpful if someone has moved the paper while the printer was idle.

If the backend will never print any burst pages, the following information can be skipped.

The printing of burst pages is done automatically by the **burst_page** routine. The routine takes two parameters: the address of a function and the width of the header and/or trailer desired. If the function address is NULL (`#include <stdio.h>`), the routine uses the supplied function and passes the character as its single parameter.

By passing the address of a special function for output, a backend can maintain strict control of what goes to the device and when it goes to the device. For example, the **burst_pages** routine uses line-feeds to separate lines, and form-feeds to separate pages. If the device requires a carriage return to precede every line-feed, the special function can make such a translation.

The basic algorithm for synchronizing calls to the **burst_page** routine with file printing looks like this:

```
burst_page(fnaddr, width);
while (files are to be printed)
{
    burst_page(fnaddr, width);
    print the next file;
```

```

    burst_page(fnaddr, width);
}

```

Every backend should follow this structure. The line numbers are used for reference in the following explanation.

The **burst_page** routine uses the information in the status file to decide whether (and how) to print a header, a trailer, some feed pages, or an aligning form-feed. The status file is set up by the **qdaemon**, using the information provide in the **qconfig** file. For example, if the **qconfig** file contains the line *header=group*, the call on line (2) results in a header page only if this file is used by a different user than the user who printed the previous file on this device. The **burst_page** routine when invoked on line (2) makes that test and either prints the header or returns. Similarly, line (3) either prints a trailer or does nothing.

With the exception of line (1), which may appear to extraneous, the algorithm is simple. This first call is necessary because **qdaemon** does not ask the backend to print a **group** trailer until it knows positively that there are no more files for a particular user. It cannot know this fact until either the first file for the next user is ready to be printed or there are no more files for this device. In the first case, **qdaemon** appends the trailer request for the previous user to the file request for the current one. Line (1) prints the trailer for the previous user if the **trailer=group** option has been selected; otherwise, it does nothing. In the second case, the backend is invoked with no file parameter at all. In this case, line (1) prints both a trailer and feed pages (assuming **qconfig** requests them), the **while** test fails, and the backend exits.

The **burst_page** routine assumes that the printer is at the top of the page, and it prints a form-feed at the end of its header or trailer to leave the printer in the same state. Backends are responsible for maintaining the position of the paper. The **align** option is useful only for device like continuous-form daisy-wheel printers, where it is possible for the printer paper to be out of alignment after a job is removed.

The **burst_page** routine should be enough for most friendly backends. If it is not, the library provides a set of routine at a lower level that should prove helpful for generating burst pages. There is a group of routines that return information from the status file, and two other routines that print headers and trailers, respectively.

Functions in the first group take no parameters; the following describes their actions:

- get_align** Returns **TRUE** or **FALSE**, telling whether an alignment form-feed is to be printed, assuming **get_newuser()** is **TRUE** and **get_endgroup()** is **FALSE**.
- get_endgroup** Returns **TRUE** or **FALSE**, telling whether this the end of a group of files for the same user.
- get_feed** Returns the number of feed pages to be printed, assuming **get_endgroup()** is **TRUE** and **get_newuser()** is **FALSE**.
- get_from** Returns the name of the person that made the print request.

get_header

Returns **NEVER**, **ALWAYS**, or **GROUP** (`#include <IN/backend.h>`).

get_lastuser

Returns the name of the previous user, assuming `get_endgroup()` is **TRUE**.

get_moddate

Returns a string showing the modification date of the file.

get_newuser

Returns **TRUE** or **FALSE**, telling whether this is the beginning of a group of files for a new user.

get_nodeid

Returns the node ID.

get_qdate Returns a string showing the date that the request was queued.

get_title Returns the title of the job being printed.

get_to Returns the name of the person for whom the job is intended.

get_trailer Returns **NEVER**, **ALWAYS**, or **GROUP**.

In addition, there is a routine `put_header(fnaddr, width)`, that prints a header with no following form-feed, returning the number of lines printed, and a routine, `put_trailer(user,fnaddr,width)`, that prints a trailer for **user**, again with no following form-feed, and returns the number of lines printed. The **fnaddr** and **width** parameters work like the same parameters in the **burst_page** function previously stated.

It should be emphasized that the auxiliary functions should not be necessary for most backends. The **burst_page()** routine handles all tasks required when it is called as described in the previous algorithm.

Extra Copies

The user can request that extra copies of a file be printed with the **print -nc** command. The `print -nc = 5 filename` command prints 5 copies of a file.

The **print** program passes the **-nc** information to the **qdaemon** process, which puts it into the status file. Backends should get the information by calling the `get_copies()` routine, which returns the total number of copies desired.

Job Status Information

The **print-q** command displays information about currently running jobs, including its originator, its title, the number of pages to be printed, and the percentage completed. All this information comes from the status file. Most of the information is set up by the **qdaemon** process when the backend is first invoked, except the **pages printed** and **percent done** fields, which must be filled in by the backend itself.

To provide this information, the backend should periodically call **log-progress(pages, percent)**, which writes the two numbers in the appropriate place in the file. The backend is free to call this routine as frequently or infrequently as desired; once at the end of each page is recommended.

Charge for the Job

Whenever a backend completes a job, the **qdaemon** process reads the status file for a charge. If the **qconfig** file has been set up appropriately, the charge is written to a file that is eventually processed by the accounting programs, resulting in a bill (real or imaginary) for the user issuing the print request.

The backend passes the charge back to the **qdaemon** process with the routine **log-charge(charge)**, where *charge* is a long integer. The backend should certainly call this routine on exit. It should also call the routine along with **log-progress** while printing the job. Otherwise, if the job is canceled, no charge will be made for the pages printed up to that point.

The charge is interpreted by all current accounting programs as the number of pages printed. However, a backend might decide that one page on its device is worth two or three normal pages (or some fraction) and set the charge accordingly.

Exit Codes

When a backend exists, the **qdaemon** process looks at its exit code for information about whether the job was completed successfully, whether the device is still usable, and so on. Therefore, it is important that backends use the same convention for their exit codes. The backend should use **#include <IN/standard.h>** for the values of the codes mentioned here.

The permissible exit codes are:

EXITOK No problems were encountered.

EXITBAD

The parameters were bad in some way. That is, a flag was unrecognizable or illegal, a file could not be opened, and so on. The **qdaemon** process notifies the user, throws out the job request, and continues sending jobs to the device.

EXITERROR

The backend could not finish printing the job and that it wants another chance. The **qdaemon** process restarts the same job (from the beginning) on the same device. The **qdaemon** process enforces a limit on the number of times that the job will be restarted.

EXITFATAL

The job could not be finished because of a problem in the device that requires manual intervention. The **qdaemon** process sets the state of the device (displayed by **print -q**) to OFF, sends a message to the console, and does not run any further jobs on that device until someone has explicitly set its state to ON again (with a **print -du**).

EXITSIGNAL

The backend was interrupted by a **SIGTERM** signal (**#include <signal.h>**).

Return Error Messages

If the backend cannot run a job (that is if it exits **EXITBAD** or **EXITFATAL**), it should send a message to the **qdaemon** process explaining the problem. The **qdaemon** process passes the message to the user, and, for **EXITFATAL** prints it on the the console.

The message should be sent with the **log_message** routine, which takes parameters in the style of **printf**:

```
log_message("cannot open file %s; error return %d\n",
            filename, erret);
```

The message cannot be longer than **MAXMSG** (**#include <IN/backend.h>**) bytes.

Set State to WAITING

The **print -q** command displays the status of a particular device. One of the entries in the table that is displayed shows whether the device is **READY**, **RUNNING**, **WAITING**, or **OFF**. This information is taken from the status file.

Normally, the **qdaemon** process keeps the status file updated, and a backend need never worry about it. However, some backends may want to explicitly set the state to **WAITING** (**#include <IN/backend.h>**) if they can no longer send output to the device, and set it back to **RUNNING** when output resumes. For example, a backend that paused at the end of each page, waiting for the user to load the next page and type a **RETURN**, might want to set the status to **WAITING** during this time.

The **log_status(status)** routine can be used to change the status of the job from **RUNNING** to **WAITING** and back again. The parameter is the new status.

Terminate on Receipt of SIGTERM

When a user cancels a running job with **print-ca**, the **print** command passes the request to the **qdaemon** process. Therefore, in order for cancellation to work, the backend must terminate soon after receipt of the signal. There are two ways to comply with this requirement.

First, the backend cannot do anything special about SIGTERM, in which case the signal kills the backend process immediately. This option is the simplest, but does not allow the backend to do any cleanup (reset line speeds, put paper at top-of-form, hang up the phone) before it terminates.

Second, the backend can catch SIGTERM, carry out whatever cleanup tasks are required, and exit **EXIT SIGNAL** (**#include <IN/standard.h>**). The special exit code tells the **qdaemon** process that the job was canceled.

Backends that decide to catch SIGTERM should exit very soon after receipt of the signal. If the cleanup code is too long, or if it can wait indefinitely (for terminal to open, for a device to respond, and so on), the backend is not friendly.

Backend Routines in libqb

The following is a list of backend routines available using the **ld** or **cc** command-line option **-lqb**.

```
burst_page(fnaddr,width)
    int (*fnaddr)();
```

```
get_align()
```

```
get_copies()
```

```
get_endgroup()
```

```
get_feed()
```

```
char *
get_from()
```

```
get_header()
```

```
char *
get_lastuser()
```

```
char *  
get_moddate()  
  
get_newuser()  
  
char *  
get_nodeid()  
  
char *  
get_qdate()  
  
char *  
get_title()  
  
char *  
get_to()  
  
get_trailer  
  
log_charge(charge)  
    long charge;  
  
log_init()  
  
log_message( . . . )  
  
log_progress(pages,percent)  
  
log_status(status)  
  
put_header(fnaddr, width)  
    int(*fnaddr)();  
  
put_trailer(user, fnaddr, width)  
    char *user;  
    int (*fnaddr)();
```

Appendix C. Porting DOS 3.0 Applications

This section is intended for experienced DOS 3.0 programmers who desire to port applications to the RT system using the AIX operating system. This section directs you to the proper section of RT documentation that contains detailed information required to port an application.

High-Level Languages

The RT has several high-level languages that are designed to allow application migration and portability or both with minimal changes to the application. The languages provided are Basic, Pascal, Fortran, and C. The Basic and Pascal languages contain the same syntax and semantics as the PC DOS versions of these languages, in addition to extensions to the language in order to take advantage of the RT architectural features. For a complete description of the languages, their extensions, and any differences between them and the PC DOS versions of the language, see *BASIC Language Reference* and *Pascal Compiler Language Reference*.

DOS File System

The DOS file system and the RT native file systems are different. However, a complete set of commands and run-time subroutines are provided for the application developer to mask these file system differences. See the **dosdel**, **dosdir**, **dosread**, **doswrite** commands in *AIX Operating System Commands Reference*, for a complete description of the commands that manipulate a DOS file system. The RT system provides a complete set of run-time subroutines that allow an application to deal with both DOS and native RT file systems. See "DOS services library," for a description of the DOS library and the DOS RT subroutine calls. See Chapter 2, "System Calls and Subroutines," for a complete description of each file system subroutine.

Note: These routines are intended to handle ASCII files in a transparent fashion. The application must handle binary data within a file.

DOS Function Calls

Most DOS function calls map to equivalent AIX operating system functions in a very straightforward way. The following is a detailed listing of DOS function calls and the equivalent AIX operating system functions that provide the same services.

Table of DOS 3.0 Function Calls

Function Calls	AIX Equivalent Function	Notes
00H Program terminate	exit	See "hft" on page 5-39.
01H Keyboard input	KSR echo, getc	
02H Display output	putc	
03H Auxiliary input	read	See "tty" on page 5-153.
04H Auxiliary output	write	See "tty" on page 5-153.
05H Printer output	write	See "lp" on page 5-117.
06H Direct console I/O	getc, putc	
07H Direct console input without echo	getchar	
08H Console input without echo	getchar	
09H Print string	puts	
0AH Buffered keyboard input	gets	
0BH Check standard input status	ioctl	See "hft" on page 5-39.
0CH Clear keyboard buffer, invoke a keyboard function	ioctl	See "hft" on page 5-39.
0DH Disk reset	sync	
0EH Select disk	Set environment variable	See "DOS services library" on page 2-153.

Function Calls	AIX Equivalent Function	Notes
0FH Open file	mapped	All FCB functions should be mapped to the equivalent file handling functions.
10H Close file	mapped	All FCB functions should be mapped to the equivalent file handling functions.
11H Search for first entry	mapped	All FCB functions should be mapped to the equivalent file handling functions.
12H Search for next entry	mapped	All FCB functions should be mapped to the equivalent file handling functions.
13H Delete file	mapped	All FCB functions should be mapped to the equivalent file handling functions.
14H Sequential read	mapped	All FCB functions should be mapped to the equivalent file handling functions.
15H Sequential write	mapped	All FCB functions should be mapped to the equivalent file handling functions.
16H Create file	mapped	All FCB functions should be mapped to the equivalent file handling functions.
17H Rename file	mapped	All FCB functions should be mapped to the equivalent file handling functions.
18H Used internally by DOS	DOS	
19H Current disk	dospwd	
1AH Set disk transfer address	N/A	
1BH Allocation table information	dosstat	
1CH Allocation table information for specific device	dosustat	

Function Calls	AIX Equivalent Function	Notes
1DH Used internally by DOS	N/A	
1EH Used internally by DOS	N/A	
1FH Used internally by DOS	N/A	
20H Used internally by DOS	N/A	
21H Random read	mapped	All FCB functions should be mapped to the equivalent file handling functions.
22H Random write	mapped	All FCB functions should be mapped to the equivalent file handling functions.
23H File size	mapped	All FCB functions should be mapped to the equivalent file handling functions.
24H Set relative record field	mapped	All FCB functions should be mapped to the equivalent file handling functions.
25H Set interrupt vector	N/A	Part of system configuration.
26H Create new program segment	exec, dosexecvec	
27H Random block read	mapped	All FCB functions should be mapped to the equivalent file handling functions.
28H Random block write	mapped	All FCB functions should be mapped to the equivalent file handling functions.
29H Parse filename	N/A	Not useful for path names.
2AH Get date	time	
2BH Set date	stime	Must have superuser authority.
2CH Get time	time	Must have superuser authority.
2DH Set time	stime	
2EH Set/reset verify switch	ioctl	See "hd" on page 5-36.

Function Calls	AIX Equivalent Function	Notes
2FH Get disk transfer address	N/A	
30H Get DOS version number	uname	
31H Terminate process and remain resident	N/A	See "exec: execl, execv, execl, execve, execlp, execvp" on page 2-226, "fork" on page 2-293, and "signal" on page 2-750.
32H Used internally by DOS	N/A	
33H Ctrl-Break check	N/A	Break key generates signal to process.
34H Used internally by DOS	N/A	
35H Get vector	N/A	
36H Get disk free space	ustat, dosustatl	
37H Used internally by DOS	N/A	
38H Set or get country dependent information	N/A	
39H Create subdirectory (MKDIR)	mkdir, dosmkdir	
3AH Remove subdirectory (RMDIR)	rm, dosrmdir	
3BH Change current directory (CHDIR)	chdir, doschdir	
3CH Create a file (CREAT)	creat, doscreate	
3DH Open a file	open, dosopen	
3EH Close a file handle	close, dosclose	
3FH Read from a file or device	read, dosread	
40H Write to a file or device	write, doswrite	
41H Delete a file from a specified directory (UNLINK)	unlink, dosunlink	
42H Move file read/write pointer (LSEEK)	lseek, doslseek	

Function Calls	AIX Equivalent Function	Notes
43H Change file mode (CHMOD)	chmod, doschmod	
44H I/O control for devices (IOCTL)	ioctl	
45H Duplicate a file handle (DUP)	dup, dosdup	
46H Force a duplicate of a file handle (FORCDUP)	fcntl	
47H Get current directory	getcwd, dospwd	
48H Allocate memory	malloc	
49H Free allocated memory	free	
4AH Modify allocated memory blocks (SETBLOCK)	realloc	
4BH Load or execute a program (EXEC)	exec, dosexecvec	
4CH Terminate a process (EXIT)	exit	
4DH Get return code of a sub-process (WAIT)	wait	
4EH Find first matching file (FIND FIRST)	dosfirst	
4FH Find next machine file	dosnext	
50H Used internally by DOS	N/A	
51H Used internally by DOS	N/A	
52H Used internally by DOS	N/A	
53H Used internally by DOS	N/A	
54H Get verify setting	ioctl	See "hd" on page 5-36.
55H Used internally by DOS	N/A	
56H Rename a file	dosrename	
57H Get/set a file's date and time	utimec, dostatc, dostouch	

Function Calls	AIX Equivalent Function	Notes
58H Used internally by DOS	N/A	
59H Get extended error	errno, doserrno	
5AH Create temporary file	mktemp, dosmktemp	
5BH Create new file	N/A	
5CH Lock/unlock file access	lockf	
5DH Used internally by DOS	N/A	
5EH Used internally by DOS	N/A	
5FH Used internally by DOS	N/A	
60H Used internally by DOS	N/A	
61H Used internally by DOS	N/A	
62H Get PSP address	N/A	

RT Hardware Access

Normal access to the RT system hardware devices is by means of special files (see chapter 6 in this book). However, it is possible for the applications to gain direct control of the hardware display to perform high function graphic applications. To use the display in this fashion, see "Monitor Mode (MOM) Output" on page 5-90. An application can also gain direct access to the hardware I/O or to manipulate a device that is unknown to the system (see "bus" on page 5-19 and *VRM Programming Support*).

For a complete description of the hardware and I/O devices supported by RT, see *Hardware Technical Reference*.

Differences Between IBM Personal Computer AT Processor and 032 Microprocessor

There are hardware differences between the processors that affect application data. For example, IBM Personal Computer AT integers are 2 bytes long and are typically stored in a left-reversed fashion on media; while 032 Microprocessor integers are 4 bytes long and are stored in sequence on the media. There are also differences in the way floating-point numbers are stored. The person developing applications is responsible for understanding and handling any differences between binary data that is exchanged between IBM Personal Computer AT architecture machines and 032 Microprocessor architecture machines. For a more detailed description of the 032

Microprocessor architecture, see *Assembler Language Reference* and *Personal Computer AT Coprocessor Services Technical Reference*.

Appendix D. Component Cross-Reference

The following subroutines and subroutine libraries are packaged with the Extended Services Program:

Subroutine or Library	Page
dbminit	2-141
delete	2-141
fetch	2-141
firstkey	2-141
libdbm.a	2-141
libdos.a	2-153
libprint.a	3-171
nextkey	2-141
store	2-141

Figure D-1. Extended Services Subroutines

The following subroutines and subroutine libraries are packaged with the Multi-User Services Program:

Subroutine or Library	Page
arc	3-171
circle	3-171
closepl	3-171
cont	3-171
erase	3-171
label	3-171

Figure D-2 (Part 1 of 2). Multi-User Services Subroutines

Subroutine or Library	Page
libgsl.a	6-1
libplot.a	3-171
libprint.a	3-171
lib300.a	3-171
lib300s.a	3-171
lib300S.a	3-171
lib4014.a	3-171
lib450.a	3-171
Sockets Library (libc.a)	2-770
line	3-171
linemod	3-171
move	3-171
openpl	3-171
point	3-171
space	3-171

Figure D-2 (Part 2 of 2). Multi-User Services Subroutines

All other system calls, subroutines, and subroutine libraries discussed in this book are packaged with the IBM RT AIX Operating System licensed program.

Appendix E. Glossary

access. To obtain data from or put data in storage.

access permission. A group of designations that determine who can access a particular AIX file and how the user may access the file.

account. The log in directory and other information that give a user access to the system.

activity manager. A collection of system-supplied tasks allowing users to manage their activities. Provides the ability to list current activities (Activity List) and to begin, cancel, hide, and activate activities.

All Points Addressable (APA) display. A display that allows each pel to be individually addressed. An APA display allows for images to be displayed that are not made up of images predefined in character boxes. Contrast with *character display*.

allocate. To assign a resource, such as a disk file or a diskette file, to perform a specific task.

alphabetic. Pertaining to a set of letters a through z.

alphanumeric character. Consisting of letters, numbers and often other symbols, such as punctuation marks and mathematical symbols.

American National Standard Code for Information Interchange (ASCII). The code developed by ANSI for information interchange among data processing systems, data communications systems, and associated equipment. The ASCII character set consists of 7-bit control characters and symbolic characters.

American National Standards Institute. An organization sponsored by the Computer and

Business Equipment Manufacturers Association for establishing voluntary industry standards.

application. A program or group of programs that apply to a particular business area, such as the Inventory Control or the Accounts Receivable application.

application program. A program used to perform an application or part of an application.

argument. Numbers, letters, or words that change the way a command works.

ASCII. See *American National Standard Code for Information Interchange*.

attribute. A characteristic. For example, the attribute for a displayed field could be blinking.

audit. To review and examine the activities of a data processing system mainly to test the adequacy and effectiveness of procedures for data privacy and data integrity.

audit bin. A file containing unprocessed audit records.

audit class. A list of events that define which action taken on a system are recorded for a user. They are defined by the system administrator in the user database.

audit event. An action (such as a command or access) taken on a system, which can be recorded by the system.

audit trail. A collection of security-compromising events recorded in the order in which they occurred.

auto carrier return. The system function that places carrier returns automatically within the text and on the display. This is accomplished by moving whole words that exceed the line end zone to the next line.

backend. The program that sends output to a particular device. There are two types of backends: friendly and unfriendly.

background process. (1) A process that does not require operator intervention that can be run by the computer while the work station is used to do other work. (2) A mode of program execution in which the shell does not wait for program completion before prompting the user for another command.

backup copy. A copy, usually of a file or group of files, that is kept in case the original file or files are unintentionally changed or destroyed.

backup diskette. A diskette containing information copied from a fixed disk or from another diskette. It is used in case the original information becomes unusable.

bad block. A portion of a disk that can never be used reliably.

base address. The beginning address for resolving symbolic references to locations in storage.

base name. The last element to the right of a full path name. A filename specified without its parent directories.

batch printing. Queueing one or more documents to print as a separate job. The operator can type or revise additional documents at the same time. This is a background process.

batch processing. A processing method in which a program or programs process records with little or no operator action. This is a background process. Contrast with *interactive processing*.

binary. (1) Pertaining to a system of numbers to the base two; the binary digits are 0 and 1. (2) Involving a choice of two conditions, such as on-off or yes-no.

bit. Either of the binary digits 0 or 1 used in computers to store information. See also *byte*.

block. (1) A group of records that is recorded or processed as a unit. Same as *physical record*. (2) In data communications, a group of records

that is recorded, processed, or sent as a unit. (3) A block is 512 bytes long. (4) A logical block is 2048 bytes long.

block file. A file listing the usage of blocks on a disk.

block special file. A special file that provides access to an input or output device is capable of supporting a file system. See also *character special file*.

bootstrap. A small program that loads larger programs during system initialization.

branch. In a computer program an instruction that selects one of two or more alternative sets of instructions. A conditional branch occurs only when a specified condition is met.

breakpoint. A place in a computer program, usually specified by an instruction, where execution may be interrupted by external intervention or by a monitor program.

buffer. (1) A temporary storage unit, especially one that accepts information at one rate and delivers it at another rate. (2) An area of storage, temporarily reserved for performing input or output, into which data is read, or from which data is written.

burst pages. On continuous-form paper, pages of output that can be separated at the perforations.

byte. The amount of storage required to represent one character; a byte is 8 bits.

call. (1) To activate a program or procedure at its entry point. Compare with *load*.

callouts. An AIX kernel parameter establishing the maximum number of scheduled activities that can be pending simultaneously.

cancel. To end a task before it is completed.

carrier return. (1) In text data, the action causing line ending formatting to be performed at the current cursor location followed by a line advance of the cursor. Equivalent to the carriage return of

a typewriter. (2) A keystroke generally indicating the end of a command line.

case sensitive. Able to distinguish between uppercase and lowercase letters.

character. A letter, digit, or other symbol.

character display. A display that uses a character generator to display predefined character boxes of images (characters) on the screen. This kind of display cannot address the screen any less than one character box at a time. Contrast with *All Points Addressable display*.

character key. A keyboard key that allows the user to enter the character shown on the key. Compare with *function keys*.

character position. On a display, each location that a character or symbol can occupy.

character set. A group of characters used for a specific reason; for example, the set of characters a printer can print or a keyboard can support.

character special file. A special file that provides access to an input or output device. The character interface is used for devices that do not use block I/O. See also *block special file*.

character string. A sequence of consecutive characters.

character variable. The name of a character data item whose value may be assigned or changed while the program is running.

child. (1) Pertaining to a secured resource, either a file or library, that uses the user list of a parent resource. A child resource can have only one parent resource. (2) In the AIX Operating System, child is a *process* spawned by a parent process that shares resources of parent process. Contrast with *parent*.

C language. A general purpose programming language that is the primary language of the AIX Operating System.

class. Pertaining to the I/O characteristics of a device. AIX devices are classified as block or character.

close. (1) To end an activity and remove that window from the display.

code. (1) Instructions for the computer. (2) To write instructions for the computer; to *program*. (3) A representation of a condition, such as an error code.

code page. In AIX, arrays of code points representing characters that establish ordinal sequence (numeric order) of characters. AIX uses 256-character code pages. Code page P0 consists of 1-byte characters that represent the ASCII, ISO, and EBCDIC character sets and additional characters and symbols. Lower code page P0 (0-127 ordinal) is the ASCII character set. Additional code pages consist of code points for 2-byte character representations.

code point. A 1- or 2-byte representation of a character. A byte can contain a single-shifted bit that indicates that the second byte is a part of the same code point. In AIX, a byte can contain a single-shifted bit that indicates the code page of the character. Also in AIX, the second byte (only byte in the case of a 1-byte character) places the character in the code page array.

code segment. See *segment*.

collating sequence. The sequence in which characters are ordered within the computer for sorting, combining, or comparing.

collation. The process of character and string sorting based on alphabetical order, and, in AIX, on equivalence class. Japanese Language Support does not use equivalence classes.

color display. A display device capable of displaying more than two colors and the shades produced via the two colors, as opposed to a monochrome display.

column. A vertical arrangement of text or numbers.

column headings. Text appearing near the top of columns of data for the purpose of identifying or titling.

command. A request to perform an operation or run a program. When parameters, arguments, flags, or other operands are associated with a command, the resulting character string is a single command.

command interpreter. A program that sends instructions to the kernel; also called an interface.

command line. The area of the screen where commands are displayed as they are typed.

command line editing keys. Keys for editing the command line.

command programming language. Facility that allows programming by the combination of commands rather than by writing statements in a conventional programming language.

compile. (1) To translate a program written in a high-level programming language into a machine language program. (2) The computer actions required to transform a source file into an executable object file.

compress. (1) To move files and libraries together on disk to create one continuous area of unused space. (2) In data communications, to delete a series of duplicate characters in a character string.

concatenate. (1) To link together. (2) To join two character strings.

condition. An expression in a program or procedure that can be evaluated to a value of either true or false when the program or procedure is running.

configuration. The group of machines, devices, and programs that make up a computer system. See also *system customization*.

configuration file. A file that specifies the characteristics of a system or subsystem, for example, the AIX queueing system.

consistent. Pertaining to a file system, without internal discrepancies.

console. (1) The main AIX display station. (2) A device name associated with the main AIX display station.

constant. A data item with a value that does not change. Contrast with *variable*.

context search. A search through a file whose target is a character string.

control block. A storage area used by a program to hold control information.

control commands. Commands that allow conditional or looping logic flow in DOS Services procedures.

control program. Part of the AIX Operating System system that determines the order in which basic functions should be performed.

controlled cancel. The system action that ends the job step being run, and saves any new data already created. The job that is running can continue with the next job step.

copy. The action by which the user makes a whole or partial duplicate of already existing data.

crash. An unexpected interruption of computer service, usually due to a serious hardware or software malfunction.

current directory. The directory that is active, and can be displayed with the **pwd** command.

current line. The line on which the cursor is located.

current working directory. See *current directory*.

cursor. (1) A movable symbol (such as an underline) on a display, used to indicate to the operator where the next typed character will be placed or where the next action will be directed. (2) A marker that indicates the current data access location within a file.

cursor movement keys. The directional keys used to move the cursor.

customize. To describe (to the system) the devices, programs, users, and user defaults for a particular data processing system.

cylinder. All fixed disk or diskette tracks that can be read or written without moving the disk drive or diskette drive read/write mechanism.

daemon. See *daemon process*.

daemon process. A process begun by the root or the root shell that can be stopped only by the root. Daemon processes generally provide services that must be available at all times such as sending data to a printer.

data block. See *block*.

data communications. The transmission of data between computers, or remote devices or both (usually over long distance).

data stream. All information (data and control information) transmitted over a data link.

debug. (1) To detect, locate, and correct mistakes in a program. (2) To find the cause of problems detected in software.

default. A value that is used when no alternative is specified by the operator.

default directory. The directory name supplied by the operating system if none is specified.

default drive. The drive name supplied by the operating system if none is specified.

default value. A value stored in the system that is used when no other value is specified.

delete. To remove. For example, to delete a file.

dependent work station. A work station having little or no standalone capability, that must be connected to a host or server in order to provide any meaningful capability to the user.

device. An electrical or electronic machine that is designed for a specific purpose and that attaches to

your computer, for example, a printer, plotter, disk drive, and so forth.

device driver. A program that operates a specific device, such as a printer, disk drive, or display.

device name. A name reserved by the system that refers to a specific device.

diagnostic. Pertaining to the detection and isolation of an error.

diagnostic aid. A tool (procedure, program, reference manual) used to detect and isolate a device or program malfunction or error.

diagnostic routine. A computer program that recognizes, locates, and explains either a fault in equipment or a mistake in a computer program.

digit. Any of the numerals from 0 through 9.

directory. A type of file containing the names and controlling information for other files or other directories.

disable. To make nonfunctional.

discipline. Pertaining to the order in which requests are serviced, for example, first-come-first-served (fcfs) or shortest job next (sjn).

disk I/O. Fixed-disk input and output.

diskette. A thin, flexible magnetic plate that is permanently sealed in a protective cover. It can be used to store information copies from the disk or another diskette.

diskette drive. The mechanism used to read and write information on diskettes.

display device. An output unit that gives a visual representation of data.

display screen. The part of the display device that displays information visually.

display station. A device that includes a keyboard from which an operator can send information to the system and a display screen on which an operator

can see the information sent to or received from the computer.

DOS Services prompt. The character string on the command line indicating the the system can accept a command (typically the \$ character).

DOS Services script. See *DOS Services* procedure.

DOS Services variables. Facilities of the DOS Services program for assigning variable values to constant names.

dump. (1) To copy the contents of all or part of storage, usually to an output device. (2) Data that has been dumped.

dump diskette. A diskette that contains a dump or is prepared to receive a dump.

dump formatter. Program for analyzing a dump.

EBCDIC. See *extended binary-coded decimal interchange code*.

EBCDIC character. Any one of the symbols included in the 8-bit EBCDIC set.

edit. To modify the form or format of data.

edit buffer. A temporary storage area used by an editor.

editor. A program used to enter and modify programs, text, and other types of documents and data.

emulation. Imitation; for example, when one computer imitates the characteristics of another computer.

enable. To make functional.

enter. To send information to the computer by pressing the Enter key.

entry. A single input operation on a work station.

environment. The settings for shell variables and paths set associated with each process. These variables can be modified later by the user.

equivalence class. In AIX, a grouping of characters (or character strings) that are considered equal for purposes of collation. For example, many languages place an uppercase character in the same equivalence class as its lowercase form, but other languages distinguish between accented and unaccented character forms for the purpose of collation. Japanese Language Support uses character class rather than equivalence class for collation.

error-correct backspace. An editing key that performs editing based on a cursor position; the cursor is moved one position toward the beginning of the line, the character at the new cursor location is deleted, and all characters following the cursor are moved one position toward the beginning of the line (to fill the vacancy left by the deleted element).

escape character. A character that suppresses the special meaning of one or more characters that follow.

exit value. A numeric value that a command returns to indicate whether it completed successfully. Some commands return exit values that give other information, such as whether a file exists. Shell programs can test exit values to control branching and looping.

expression. A representation of a value. For example, variables and constants appearing alone or in combination with operators.

extended binary-coded decimal interchange code (EBCDIC). A set of 256 eight-bit characters.

feature. A programming or hardware option, usually available at an extra cost.

field. (1) An area in a record or panel used to contain a particular category of data. (2) The smallest component of a record that can be referred to by a name.

FIFO. See *first-in-first-out*.

file. A collection of related data that is stored and retrieved by an assigned name.

file name. The name used by a program to identify a file. See also *label*.

filename. In DOS, that portion of the file name that precedes the extension.

file specification (filespec). The name and location of a file. A file specification consists of a drive specifier, a path name, and a file name.

file system. The collection of files and file management structures on a physical or logical mass storage device, such as a diskette or minidisk.

filetab. An AIX kernel parameter establishing the maximum number of files that can be open simultaneously.

filter. A command that reads standard input data, modifies the data, and sends it to standard output.

first-in-first-out (FIFO). A named permanent pipe. A FIFO allows two unrelated processes to exchange information using a pipe connection.

fixed disk. A flat, circular, nonremoveable plate with a magnetizable surface layer on which data can be stored by magnetic recording.

fixed-disk drive. The mechanism used to read and write information on fixed disk.

flag. A modifier that appears on a command line with the command name that defines the action of the command. Flags in the AIX Operating System almost always are preceded by a dash.

flattened character. In AIX, an ASCII character created by translating an extended character to its ASCII equivalent in appearance. Code point information is lost; the character cannot be retranslated to an extended character.

font. A family or assortment of characters of a given size and style.

foreground. A mode of program execution in which the shell waits for the program specified on the command line to complete before returning your prompt.

format. (1) A defined arrangement of such things as characters, fields, and lines, usually used for displays, printouts, or files. (2) The pattern which determines how data is recorded.

formatted diskette. A diskette on which control information for a particular computer system has been written but which may or may not contain any data.

free list. A list of available space on each file system. This is sometimes called the free-block list.

free-block list. See *free list*.

full path name. The name of any directory or file expressed as a string of directories and files beginning with the root directory.

function. A synonym for procedure. The C language treats a function as a data type that contains executable code and returns a single value to the calling function.

function keys. Keys that request actions but do not display or print characters. Included are the keys that normally produce a printed character, but when used with the code key produce a function instead. Compare with *character key*.

generation. For some remote systems, the translation of configuration information into machine language.

Gid. See *group number*.

global. Pertains to information available to more than one program or subroutine.

global action. An action having general applicability, independent of the context established by any task.

global character. The special characters * and ? that can be used in a file specification to match one or more characters. For example, placing a ? in a file specification means any character can be in that position.

global search. The process of having the system look through a document for specific characters, words, or groups of characters.

global variable. A symbol defined in one program module, but used in other independently assembled program modules.

graphic character. A character that can be displayed or printed.

group name. A name that uniquely identifies a group of users to the system.

group number (Gid). A unique number assigned to a group of related users. The group number can often be substituted in commands that take a group name as an argument.

hardware. The equipment, as opposed to the programming, of a computer system.

header. Constant text that is formatted to be in the top margin of one or more pages.

header label. A special set of records on a diskette describing the contents of the diskette.

here document. Data contained within a DOS Services program or procedure (also called *inline input*).

highlight. To emphasize an area on the display by any of several methods, such as brightening the area or reversing the color of characters within the area.

history file. A file containing a log of system actions and operator responses.

hog factor. In system accounting, an analysis of how many times each command was run, how much processor time and memory it used, and how intensive that use was.

home directory. (1) A directory associated with an individual user. (2) The user's current directory on login or after issuing the **cd** command with no argument.

I/O. See *input/output*.

ID. Identification.

IF expressions. Expressions within a procedure, used to test for a condition.

indirect block. A block containing pointers to other blocks. Indirect blocks can be single-indirect, double-indirect, or triple-indirect.

informational message. A message providing information to the operator, that does not require a response.

initial program load (IPL). The process of loading the system programs and preparing the system to run jobs. See *initialize*.

initialize. To set counters, switches, addresses, or contents of storage to zero or other starting values at the beginning of, or at prescribed points in, the operation of a computer routine.

inline input. See *here document*.

i-node. The internal structure for managing files in the system. I-nodes contain all of the information pertaining to the node, type, owner, and location of a file. A table of i-nodes is stored near the beginning of a file system.

i-number. A number specifying a particular i-node on a file system.

inodetab. An AIX kernel parameter that establishes a table in memory for storing copies of i-nodes for all active files.

input. Data to be processed.

input device. Physical devices used to provide data to a computer.

input file. A file opened by a program so that the program can read from that file.

input list. A list of variables to which values are assigned from input data.

input redirection. The specification of an input source other than the standard one.

input-output file. A file opened for input and output use.

input-output device number. A value assigned to a device driver by the guest operating system or to the virtual device by the virtual resource manager. This number uniquely identifies the device regardless of whether it is real or virtual.

input/output (I/O). Pertaining to either input, output, or both between a computer and a device.

interactive processing. A processing method in which each system user action causes response from the program or the system. Contrast with *batch processing*.

interface. A shared boundary between two or more entities. An interface might be a hardware component to link two devices together or it might be a portion of storage or registers accessed by two or more computer programs.

interleave factor. Specification of the ratio between contiguous physical blocks (on a fixed-disk) and logically contiguous blocks (as in a file).

interrupt. (1) To temporarily stop a process. (2) In data communications, to take an action at a receiving station that causes the sending station to end a transmission. (3) A signal sent by an I/O device to the processor when an error has occurred or when assistance is needed to complete I/O. An interrupt usually suspends execution of the currently executing program.

IPL. See *initial program load*.

job. (1) A unit of work to be done by a system. (2) One or more related procedures or programs grouped into a procedure.

job queue. A list, on disk, of jobs waiting to be processed by the system.

justify. To print a document with even right and left margins.

kbuffers. An AIX kernel parameter establishing the number of buffers that can be used by the kernel.

K-byte. See *kilobyte*.

kernel. The memory-resident part of the AIX Operating System containing functions needed immediately and frequently. The kernel supervises the input and output, manages and controls the hardware, and schedules the user processes for execution.

kernel parameters. Variables that specify how the kernel allocates certain system resources.

key pad. A physical grouping of keys on a keyboard (for example, numeric key pad, and cursor key pad).

keyboard. An input device consisting of various keys allowing the user to input data, control cursor and pointer locations, and to control the dialog between the user and the display station

keylock feature. A security feature in which a lock and key can be used to restrict the use of the display station.

keyword. One of the predefined words of a programming language; a reserved word.

keyword argument. One type of variable assignment that can be made on the command line.

kill. An AIX Operating System command that stops a process.

kill character. The character that is used to delete a line of characters entered after the user's prompt.

kilobyte. 1024 bytes.

kprocs. An AIX kernel parameter establishing the maximum number of processes that the kernel can run simultaneously.

label. (1) The name in the disk or diskette volume table of contents that identifies a file. See also *file name*. (2) The field of an instruction that assigns a symbolic name to the location at which the instruction begins, or such a symbolic name.

left justify. See *left-adjust*.

left margin. The area on a page between the left paper edge and the leftmost character position on the page.

left-adjust. The process of aligning lines of text at the left margin or at a tab setting such that the leftmost character in the line or field is in the leftmost position. Contrast with *right-adjust*.

library. A collection of functions, calls, subroutines, or other data.

Licensed Program (LP). Software programs that remain the property of the manufacturer, for which customers pay a license fee.

line editor. An editor that modifies the contents of a file one line at a time.

linefeed. An ASCII character that causes an output device to move forward one line.

link. A connection between an i-node and one or more file names associated with it.

literal. A symbol or a quantity in a source program that is itself data, rather than a reference to data.

load. (1) To move data or programs into storage. (2) To place a diskette into a diskette drive, or a magazine into a diskette magazine drive. (3) To insert paper into a printer.

loader. A program that reads run files into main storage, thus preparing them for execution.

local. Pertaining to a device directly connected to your system without the use of a communications line. Contrast with *remote*.

log. To record; for example, to log all messages on the system printer. A list of this type is called a log, such as an error log.

log in. To begin a session at a display station.

log in shell. The program, or command interpreter, started for a user at log in.

log off. To end a session at a display station.

log out. To end a session at a display station.

logical device. A file for conducting input or output with a physical device.

loop. A sequence of instructions performed repeatedly until an ending condition is reached.

main storage. The part of the processing unit where programs are run.

maintenance system. A special version of the AIX Operating System which is loaded from diskette and used to perform system management tasks.

major device number. A system identification number for each device or type of device.

mapped files. Files on the fixed-disk that are accessed as if they are in memory.

mask. A pattern of characters that controls the keeping, deleting, or testing of portions of another pattern of characters.

matrix. An array arranged in rows and columns.

maxprocs. An AIX kernel parameter establishing the maximum number of processes that can be run simultaneously by a user.

memory. Storage on electronic chips. Examples of memory are random access memory, read only memory, or registers. See *storage*.

menu. A displayed list of items from which an operator can make a selection.

message. (1) A response from the system to inform the operator of a condition which may affect further processing of a current program. (2) Information sent from one user in a multi-user operating system to another.

minidisk. A logical division of a fixed disk.

minor device number. A number used to specify various types of information about a particular device, for example, to distinguish among several printers of the same type.

mode word. An i-node field that describes the type and state of the i-node.

modem. See *modulator-demodulator*.

modulation. Changing the frequency or size of one signal by using the frequency or size of another signal.

modulator-demodulator (modem). A device that converts data from the computer to a signal that can be transmitted on a communications line, and converts the signal received to data for the computer.

module. (1) A discrete programming unit that usually performs a specific task or set of tasks. Modules are subroutines and calling programs that are assembled separately, then linked to make a complete program. (2) See *load module*.

mount. To make a file system accessible.

mountab. An AIX kernel parameter establishing the maximum number of file systems that can be mounted simultaneously.

multiprogramming. The processing of two or more programs at the same time.

multivolume file. A diskette file occupying more than one diskette.

nest. To incorporate a structure or structures of some kind into a structure of the same kind. For example, to nest one loop (the nested loop) within another loop (the nesting loop); to nest one subroutine (the nested subroutine) within another subroutine (the nesting subroutine).

network. A collection of products connected by communication lines for information exchange between locations.

IBM AIX/RT Network File System (NFS). A network file service that allows users to share files across a network of heterogeneous machines by mounting remotely the file systems of designated servers.

new-line character. A control character that causes the print or display position to move to the first position on the next line.

null. Having no value, containing nothing.

null character (NUL). The character hex 00, used to represent the absence of a printed or displayed character.

numeric. Pertaining to any of the digits 0 through 9.

object code. Machine-executable instruction, usually generated by a compiler from source code written in a higher level language. consists of directly executable machine code. For programs that must be linked, object code consists of relocatable machine code.

octal. A base eight numbering system.

open. (1) To make a file available to a program for processing.

operating system. Software that controls the running of programs; in addition, an operating system may provide services such as resource allocation, scheduling, input/output control, and data management.

operation. A specific action (such as move, add, multiply, load) that the computer performs when requested.

operator. A symbol representing an operation to be done.

output. The result of processing data.

output devices. Physical devices used by a computer to present data to a user.

output file. A file that is opened by a program so that the program can write to that file.

output redirection. The specification of an output destination other than the standard one.

override. (1) A parameter or value that replaces a previous parameter or value. (2) To replace a parameter or value.

overwrite. To write output into a storage or file space that is already occupied by data.

owner. The user who has the highest level of access authority to a data object or action, as defined by the object or action.

pad. To fill unused positions in a field with dummy data, usually zeros or blanks.

page. A block of instructions, data, or both.

page space minidisk. The area on a fixed disk that temporarily stores instructions or data currently being run. See also *minidisk*.

pagination. The process of adjusting text to fit within margins and/or page boundaries.

paging. The action of transferring instructions, data, or both between real storage and external page storage.

parallel processing. The condition in which multiple tasks are being performed simultaneously within the same activity.

parameter. Information that the user supplies to a panel, command, or function.

parent. Pertaining to a secured resource, either a file or library, whose user list is shared with one or more other files or libraries. Contrast with *child*.

parent directory. The directory one level above the current directory.

partition. See *minidisk*.

password. A string of characters that, when entered along with a user identification, allows an operator to sign on to the system.

password security. A program product option that helps prevent the unauthorized use of a display station, by checking the password entered by each operator at sign-on.

path name. See *full path name* and *relative path name*.

pattern-matching character. Special characters such as * or ? that can be used in search patterns. Some used in a file specification to match one or more characters. For example, placing a ? in a file

specification means any character can be in that position. Pattern-matching characters are also called wildcards.

permission code. A three-digit octal code, or a nine-letter alphabetic code, indicating the access permissions. The access permissions are read, write, and execute.

permission field. One of the three-character fields within the permissions column of a directory listing indicating the read, write, and run permissions for the file or directory owner, group, and all others.

phase. One of several stages file system checking and repair performed by the **fsck** command.

physical device. See *device*.

physical file. An indexed file containing data for which one or more alternative indexes have been created.

physical record. (1) A group of records recorded or processed as a unit. Same as *block*. (2) A unit of data moved into or out of the computer.

PID. See *process ID*.

pipe. To direct the data so that the output from one process becomes the input to another process.

pipeline. A direct, one-way connection between two or more processes.

pitch. A unit of width of typewriter type, based on the number of times a letter can be set in a linear inch. For example, 10-pitch type has 10 characters per inch.

platen. The support mechanism for paper on a printer, commonly cylindrical, against which printing mechanisms strike to produce an impression.

pointer. A logical connection between physical blocks.

port. (1) To make the programming changes necessary to allow a program that runs on one type of computer to run on another type of computer.

(2) An access point for data input to or data output from a computer system. See *connector*.

position. The location of a character in a series, as in a record, a displayed message, or a computer printout.

positional parameter. A DOS Services facility for assigning values from the command line to variables in a program.

print queue. A file containing a list of the names of files waiting to be printed.

printout. Information from the computer produced by a printer.

priority. The relative ranking of items. For example, a job with high priority in the job queue will be run before one with medium or low priority.

priority number. A number that establishes the relative priority of printer requests.

privileged user. The account with superuser authority.

problem determination. The process of identifying why the system is not working. Often this process identifies programs, equipment, data communications facilities, or user errors as the source of the problem.

problem determination procedure. A prescribed sequence of steps aimed at recovery from, or circumvention of, problem conditions.

procedure. See *shell procedure*.

process. (1) A sequence of actions required to produce a desired result. (2) An entity receiving a portion of the processor's time for executing a program. (3) An activity within the system begun by entering a command, running a shell program, or being started by another process.

process accounting. An analysis of the use each process makes of the processing unit, memory, and I/O resources.

process ID (PID). A unique number assigned to a process that is running.

profile. (1) A file containing customized settings for a system or user (2) Data describing the significant features of a user, program, or device.

program. A file containing a set of instructions conforming to a particular programming language syntax.

prompt. A displayed request for information or operator action.

propagation time. The time necessary for a signal to travel from one point on a communications line to another.

qdaemon. The daemon process that maintains a list of outstanding jobs and sends them to the specified device at the appropriate time.

queue. A line or list formed by items waiting to be processed.

queued message. A message from the system that is added to a list of messages stored in a file for viewing by the user at a later time. This is in contrast to a message that is sent directly to the screen for the user to see immediately.

quit. A key, command, or action that tells the system to return to a previous state or stop a process.

quote. To mask the special meaning of certain characters; to cause them to be taken literally.

random access. An access mode in which records can be read from, written to, or removed from a file in any order.

raw socket. A socket of the type **SOCK_RAW** used to allow direct access to lower-level protocols.

readonly. Pertaining to file system mounting, a condition that allows data to be read, but not modified.

recovery procedure. (1) An action performed by the operator when an error message appears on the display screen. Usually, this action permits the program to continue or permits the operator to run the next job. (2) The method of returning the

system to the point where a major system error occurred and running the recent critical jobs again.

redirect. To divert data from a process to a file or device to which it would not normally go.

reference count. In an i-node, a record of the total number of directory entries that refer to the i-node.

relational expression. A logical statement describing the relationship (such as greater than or equal) of two arithmetic expressions or data items.

relational operator. The reserved words or symbols used to express a relational condition or a relational expression.

relative address. An address specified relative to the address of a symbol. When a program is relocated, the addresses themselves will change, but the specification of relative addresses remains the same.

relative addressing. A means of addressing instructions and data areas by designating their locations relative to some symbol.

relative path name. The name of a directory or file expressed as a sequence of directories followed by a file name, beginning from the current directory.

remote. Pertaining to a system or device that is connected to your system through a communications line. Contrast with *local*.

Remote Procedure Call (RPC). An interface that allow remote communication with function call semantics. RPC provides a standard protocol for initiating and controlling processes on remote systems.

reserved character. A character or symbol that has a special (non-literal) meaning unless quoted.

reserved word. A word that is defined in a programming language for a special purpose, and that must not appear as a user-declared identifier.

reset. To return a device or circuit to a clear state.

restore. To return to an original value or image. For example, to restore a library from diskette.

right justify. See *right-adjust*.

right margin. The area on a page between the last text character and the right upper edge.

right-adjust. To place or move an entry in a field so that the rightmost character of the field is in the rightmost position. Contrast with *left-adjust*.

root. Another name sometimes used for superuser.

root directory. The top level of a tree-structured directory system.

root file system. The basic AIX Operating System file system, which contains operating system files and onto which other file systems can be mounted. The root file system is the file system that contains the files that are run to start the system running.

routine. A set of statements in a program causing the system to perform an operation or a series of related operations.

run. To cause a program, utility, or other machine function to be performed.

run-time environment. A collection of subroutines and shell variables that provide commonly used functions and information for system components.

scratch file. A file, usually used as a work file, that exists until the program that uses it ends.

screen. See *display screen*.

scroll. To move information vertically or horizontally to bring into view information that is outside the display screen boundaries.

sector. (1) An area on a disk track or a diskette track reserved to record information. (2) The smallest amount of information that can be written to or read from a disk or diskette during a single read or write operation.

security. The protection of data, system operations, and devices from accidental or intentional ruin, damage, or exposure.

segment. A contiguous area of virtual storage allocated to a job or system task. A program segment can be run by itself, even if the whole program is not in main storage.

separator. A character used to separate parts of a command or file.

sequential access. An access method in which records are read from, written to, or removed from a file based on the logical order of the records in the file.

session records. In the accounting system, a record of time connected and line usage for connected display stations, produced from log in and log out records.

set flags. Flags that can be put into effect with the DOS Services set command.

shared printer. A printer that is used by more than one work station.

shell. See *shell program*.

shell procedure. A series of commands combined in a file that carry out a particular function when the file is run or when the file is specified as an argument to the sh command. Shell procedures are frequently called shell scripts.

shell program. A program that accepts and interprets commands for the operating system (there is an AIX shell program and a DOS shell program).

size field. In an i-node, a field that indicates the size, in bytes, of the file associated with the i-node.

software. Programs.

sort. To rearrange some or all of a group of items based upon the contents or characteristics of those items.

source diskette. The diskette containing data to be copied, compared, restored, or backed up.

source program. A set of instructions written in a programming language, that must be translated to machine language compiled before the program can be run.

special character. A character other than an alphabetic or numeric character. For example; *, +, and % are special characters.

special file. Special files are used in the AIX system to provide an interface to input/output devices. There is at least one special file for each device connected to the computer. Contrast with *directory* and *file*. See also *block special file* and *character special file*.

spool files. Files used in the transmission of data among devices.

standalone DOS Services. A limited version of the DOS Services program used for system maintenance.

standalone work station. A work station that can be used to preform tasks independent of (without being connected to) other resources such as servers or host systems.

standard error. The place where many programs place error messages.

standard input. The primary source of data going into a command. Standard input comes from the keyboard unless redirection or piping is used, in which case standard input can be from a file or the output from another command.

standard output. The primary destination of data coming from a command. Standard output goes to the display unless redirection or piping is used, in which case standard output can be to a file or another command.

stanza. A group of lines in a file that together have a common function. Stanzas are usually separated by blank lines, and each stanza has a name.

statement. An instruction in a program or procedure.

status. (1) The current condition or state of a program or device. For example, the status of a printer. (2) The condition of the hardware or software, usually represented in a status code.

storage. (1) The location of saved information. (2) In contrast to memory, the saving of information on physical devices such as disk or tape. See *memory*.

storage device. A device for storing and/or retrieving data.

string. A linear sequence of entities such as characters or physical elements. Examples of strings are alphabetic string, binary element string, bit string, character string, search string, and symbol string.

su. See *superuser*.

subdirectory. A directory contained within another directory in the file system hierarchy.

subprogram. A program invoked by another program, such as a subshell.

subroutine. (1) A sequenced set of statements that may be used in one or more computer programs and at one or more points in a computer program. (2) A routine that can be part of another routine.

subscript. An integer or variable whose value refers to a particular element in a table or an array.

subshell. An instance of the DOS Services program started from an existing DOS Services program.

substring. A part of a character string.

subsystem. A secondary or subordinate system, usually capable of operating independently of, or synchronously with, a controlling system.

superblock. The most critical part of the file system containing information about every allocation or deallocation of a block in the file system.

superuser (su). The user who can operate without the restrictions designed to prevent data loss or damage to the system (User ID 0).

superuser authority. The unrestricted ability to access and modify any part of the operating system

associated with the user who manages the system. The authority obtained when one logs in as **root**.

system. The computer and its associated devices and programs.

system call. A request by an active process for a service by the system kernel.

system customization. A process of specifying the devices, programs, and users for a particular data processing system.

system date. The date assigned by the system user during setup and maintained by the system.

system dump. A copy of memory from all active programs (and their associated data) whenever an error stops the system. Contrast with *task dump*.

system management. The tasks involved in maintaining the system in good working order and modifying the system to meet changing requirements.

system parameters. See *kernel parameters*.

system profile. A file containing the default values used in system operations.

system unit. The part of the system that contains the processing unit, the disk drives, and the diskette drives.

system user. A person who uses a computer system.

target diskette. The diskette to be used to receive data from a source diskette.

task. A basic unit of work to be performed. Examples are a user task, a server task, and a processor task.

task dump. A copy of memory from a program that failed (and its associated data). Contrast with *system dump*.

terminal. An input/output device containing a keyboard and either a display device or a printer. Terminals usually are connected to a computer and allow a person to interact with the computer.

text. A type of data consisting of a set of linguistic characters (for example, alphabet, numbers, and symbols) and formatting controls.

text application. A program defined for the purpose of processing text data (for example, memos, reports, and letters).

text editing program. See *editor* and *text application*.

texttab. A kernel parameter establishing the size of the text table, in memory, that contains one entry each active shared program text segment.

trace. To record data that provides a history of events occurring in the system.

trace table. A storage area into which a record of the performance of computer program instructions is stored.

track. A circular path on the surface of a fixed disk or diskette on which information is magnetically recorded and from which recorded information is read.

trap. An unprogrammed, hardware-initiated jump to a specific address. Occurs as a result of an error or certain other conditions.

tree-structured directories. A method for connecting directories such that each directory is listed in another directory except for the root directory, which is at the top of the tree.

truncate. To shorten a field or statement to a specified length.

trusted communications path. A secure path to the system, invoked with a key sequence and used when entering or changing security-relevant information in the system. For example, when changing passwords or logging in to the system.

trusted shell. A modified command interpreter that provides a restricted environment to perform administrative tasks in a secure manner.

typematic key. A key that repeats its function multiple times when held down.

typestyle. Characters of a given size, style and design.

Uid. See *user number*.

update. An improvement for some part of the system.

user. The name associated with an account.

user account. See *account*.

user ID. See *user number*.

user name. A name that uniquely identifies a user to the system.

user number (Uid). (1) A unique number identifying an operator to the system. This string of characters limits the functions and information the operator is allowed to use. The Uid can often be substituted in commands that take a user's name as an argument.

user profile. A file containing a description of user characteristics and defaults (for example, printer assignment, formats, group ID) to be conveyed to the system while the user is signed on.

utility. A service; in programming, a program that performs a common service function.

valid. (1) Allowed. (2) True, in conforming to an appropriate standard or authority.

value. (1) In Usability Services, information selected or typed into a pop-up. (2) A set of characters or a quantity associated with a parameter or name. (3) In programming, the contents of a storage location.

variable. A name used to represent a data item whose value can change while the program is running. Contrast with *constant*.

verify. To confirm the correctness of something.

version. Information in addition to an object's name that identifies different modification levels of the same logical object.

virtual device. A device that appears to the user as a separate entity but is actually a shared portion of a real device. For example, several virtual terminals may exist simultaneously, but only one is active at any given time.

virtual machine. A functional simulation of a computer and its related devices.

virtual machine interface (VMI). A software interface between work stations and the operating system. The VMI shields operating system software from hardware changes and low-level interfaces and provides for concurrent execution of multiple virtual machines.

virtual resource manager (VRM). A set of programs that manage the hardware resources (main storage, disk storage, display stations, and printers) of the system so that these resources can be used independently of each other.

virtual resources. See *virtual resource manager*.

virtual storage. Addressable space that appears to be real storage. From virtual storage, instructions and data are mapped into real storage locations.

virtual terminal. Any of several logical equivalents of a display station available at a single physical display station.

Volume ID (Vol ID). A series of characters recorded on the diskette used to identify the diskette to the user and to the system.

VRM. See *virtual resource manager*.

wildcard. See *pattern-matching characters*.

word. A contiguous series of 32 bits (4 bytes) in storage, addressable as a unit. The address of the first byte of a word is evenly divisible by four.

work file. A file used for temporary storage of data being processed.

work station. A device at which an individual may transmit information to, or receive information from, a computer for the purpose of performing a task, for example, a display station or printer. See *programmable work station* and *dependent work station*.

working directory. See *current directory*.

wrap around. Movement of the point of reference in a file from the end of one line to the beginning of the next, or from one end of a file to the other.

XDR (eXternal Data Representation). A data definition language that represents internal machine data types in a uniform format. NFS uses XDR to function independently of machine types, operating systems and network architecture when making remote procedure calls across remote machines in the network.

Yellow Pages (YP). A network service used to administer system information, such as passwords and host names. It is installed as an option to the IBM AIX/RT Network File System.

Special Characters

- _**N**Ctoupper macro
- .forward file 3-151
- .maildelivery file 3-151
- .init.state file format 3-3
- /etc/security/audit/events 3-73
- /usr/lpp/rem/samples 2-691
- _**a**tojis subroutine 2-105
- _**C**_ prefix 2-236, 2-765, 4-64
- _**C**_func 2-236, 2-765, 4-64
- _**e**xit system call 2-233
- _**j**istoa subroutine 2-105
- _**N**Ctolower macro
- _**N**Cxcol macro 2-497
- _**N**Lxcol macro 2-497
- _**t**ojlower subroutine 2-105
- _**t**ojupper subroutine 2-105
- _**t**olower subroutine 2-105
- _**t**oupper subroutine 2-105
- _**N**Ctolower macro2 2-105
- _**N**Ctoupper macro2 2-105
- _**r**es structure 2-655

A

- a.out file 3-5
- a.out relocation 3-9
- a.out segments
 - data 3-5
 - stack 3-5
 - text 3-5
- a.out structure 3-5
- abort subroutine 2-15
- abs subroutine 2-16
- absolute path 1-26
- absolute value function 2-290

- absolute value, integer 2-16
- accept
 - socket connection 2-17
- accept socket subroutine 2-17
- access
 - exclusive 2-427
- access list
 - group 2-404, 2-724
- access system call 2-19
- access time
 - file 2-850
- access utmp file entry 2-393
- accessibility, determine file 2-19
- accounting
 - process 2-22
- accounting file structure 3-15
- accounting, process file 3-15
- acct file 3-15
- acct system call 2-22
- acos subroutine 2-764
- action
 - upon receipt of signal 2-750
- acute accent character 4-10
- add a device 2-64
- add a minidisk 2-67
- add protocol procedure 2-869
- addch subroutine 2-130, 2-242
- addressing
 - kernel mode 1-10
 - user mode 1-8
- addstr subroutine 2-130, 2-243
- Advanced Display Graphics Support Library 6-2
- Advanced Floating-Point Accelerator 2-295, 2-308
- afork flag 3-16
- AIX file system 1-18
- AIX system name
 - extended 2-839
 - get 2-839

AIX trace collector 2-827
 alarm clock
 set 2-25
 alarm system call 2-25, 2-661
 alias file, message system 3-138
 allocating free blocks 1-25
 allocation
 change data segment space 2-48
 free blocks 1-25
 i-number 1-24
 allocator, main memory 2-436
 alphabetical sort
 of an array 2-26
 alphasort subroutine 2-26
 ANSI floating point 2-295
 APC/881 2-295, 2-308, 2-315
 append
 data to a file 2-876
 append audit record 2-39
 apply configuration information 2-69
 ar file 3-18
 arc subroutine 2-549
 arccosine function 2-764
 archive file format 3-18
 archive file member structure 3-18
 archive format, cpio 3-45
 arcsine function 2-764
 arctangent function 2-764
 argc parameter 2-227
 argument list, print 2-866
 argv parameter 2-227
 array
 sort alphabetically 2-26
 ASCII character set 4-3
 ASCII controls 4-11
 ASCII facility 4-3
 ASCII to floating-point conversion 2-798
 ASCII to integer conversion 2-13
 asctime subroutine 2-120
 asin subroutine 2-764
 assembler output file 3-5
 assert subroutine 2-28
 assertion verification 2-28
 assign a DOS Services drive 2-158
 assign buffering to a stream 2-720, 2-722
 atan subroutine 2-764
 atan2 subroutine 2-764
 atof subroutine 2-798
 atoi subroutine 2-800
 atojis subroutine 2-105
 _atojis subroutine 2-105
 atol subroutine 2-800
 atomic operation 2-755
 attach
 mapped file 2-734
 shared memory segment 2-734
 attribute file 2-71
 attribute file, close 2-73
 attribute file, read stanza 2-79
 attribute files 2-75, 2-77
 attributes
 file system 3-74
 serverattach 3-204
 attributes file 3-20
 attroff subroutine 2-130
 attron subroutine 2-130
 attrset subroutine 2-130
 audit action name 3-73
 audit device 5-5
 audit event name 3-73
 audit file format 3-22
 audit record
 append 2-39
 audit state 2-36
 get for process 2-41
 set for process 2-41
 audit system
 append audit record 2-39
 audit action name 3-73
 audit event name 3-73
 disable 2-31, 2-33
 enable 2-31, 2-33
 queries state 2-36
 set state 2-36
 audit system call 2-31
 audit trail file 3-22
 auditbin system call 2-33
 auditevents system call 2-36
 auditlog system call 2-39
 auditproc system call 2-41
 automatic new line mode (AUTONL) 5-86
 AUTONL mode 5-86

a64l subroutine 2-13

B

backend

- burst pages B-3
- exit codes B-6
- extra print copies B-5
- job charge B-6
- job status information B-6
- return error messages B-7
- routines in libqb B-8
- SIGTERM terminate B-8
- waiting state B-7

backends B-1

backup file 3-24

baudrate subroutine 2-130

bcmp subroutine 2-52

bcopy subroutine 2-445

beep subroutine 2-130, 2-244

bessel subroutines 2-44

binary input/output 2-320

binary search 2-50

binary search trees 2-829

binary synchronous communications 5-26

bind

- name to socket 2-46

bind socket subroutine 2-46

biodd 5-7

BISYNC 5-26

block

- signal delivery 2-748

Block I/O Kernel Device Driver 5-7

block 0 layout 1-21

blocked signals

- release 2-755

blocks

- allocation of free 1-25

- data 1-24

- delayed 2-806

- free 1-24

- superblock 1-21

bootstrap 1-5, 3-3

box subroutine 2-131, 2-244

breve accent character 4-10

brk system call 2-48

BRKINT 5-136

bsc device driver 5-26

BSD

BSDLY 5-137

bsearch subroutine 2-50

BS0 5-137

BS1 5-137

buffer subsystem 1-32

buffered I/O 2-783

buffering assignment to a stream 2-720

bus

- I/O 1-33

bus special file 5-19

byte order conversion

- host to network 2-399

- network to host 2-399

byte string operations 2-52

byte swapping 2-802

bzero subroutine 2-52

C

C prefix 2-236, 2-765, 4-64

_C_func 2-236, 2-765, 4-64

caddr_t data type 4-78

call messages

- See Remote Procedure Calls (RPC)

call switch table 1-32

calling sequence ix

calloc subroutine 2-436

calls

- to devices 1-36

calls, AIX supervisor

- See system calls and subroutines

calls, function

- See kernel subroutines

- See system calls and subroutines

calls, kernel

- See kernel subroutines

- See system calls and subroutines

calls, routine

- See kernel subroutines

See system calls and subroutines
 calls, subroutine
 See kernel subroutines
 See system calls and subroutines
 calls, supervisor, AIX
 See system calls and subroutines
 calls, system
 See system calls and subroutines
 cancel sound 5-84
 caron accent character 4-10
 case
 conversion 2-105, 2-514, 2-516
 translation 2-105, 2-514
 catclose subroutine 2-54
 catgetmsg subroutine 2-56-2-57
 catgets subroutine 2-58-2-59
 catopen subroutine 2-60-2-61
 CBAUD 5-138
 cbox subroutine 2-244
 cbreak subroutine 2-131
 cc.cfg file 3-31
 cedilla accent character 4-10
 ceil subroutine 2-290
 ceiling function 2-290
 cfgabdds subroutine 2-62
 cfgadev subroutine 2-64
 cfgamni subroutine 2-67
 cfgaply subroutine 2-69
 cfgcadsz subroutine 2-71
 cfgccslf subroutine 2-73
 cfgcdlsz subroutine 2-75
 cfgcopsf subroutine 2-77
 cfgcrdsz subroutine 2-79
 cfgddev subroutine 2-81
 cfgdmni subroutine 2-84
 cfggetbp subroutine 2-86
 change
 access permissions 2-89
 current directory 2-87
 data segment space allocation 2-48
 effective root directory 2-97
 file mode 2-89
 group of a file 2-93
 owner of a file 2-93
 change current DOS Services directory 2-160
 change current DOS Services drive 2-160
 change DOS file mode 2-162
 change fonts 5-87
 change modification date of DOS file 2-196
 change priority
 of a process 2-508
 channel
 create 2-545
 character
 conversion 2-516
 locate in string 2-401
 single shift 4-9
 2-byte 4-9
 character classification 2-123
 international character support 2-500
 character code processing 5-86
 character codes 4-26
 character collation
 code point 2-520
 international character support 2-497
 character I/O 2-841
 character set
 ASCII 4-3
 character set definition 5-86
 character translation 2-105, 2-514
 character, get from stream 2-342
 characteristics, device 3-66
 characters
 international character support 2-514
 characters, nonspacing 4-10
 characters, 2-byte 4-9
 chdir system call 2-87
 check whether trace channel is enabled 2-822
 check_timers subroutine 2-670
 chgat subroutine 2-244
 child process 1-13, 2-293
 wait for termination of 2-872, 2-874
 child process times
 get 2-817
 chmod system call 2-89
 chown system call 2-93
 chownx system call 2-93
 chroot system call 2-97
 circle subroutine 2-549
 circumflex accent character 4-10
 classify characters 2-123
 clear subroutine 2-131, 2-245

clearerr macro 2-285
clearok subroutine 2-131, 2-245
client
 See Remote Procedure Calls (RPC)
CLOCAL 5-139
clock
 set alarm 2-25
clock rate 2-818
clock resolution 2-100
clock subroutine 2-100
close
 a file 2-101
 log file 2-807
 network data base 2-364
 network host data base 2-353
 network protocol data base 2-374
 network services data base 2-381
close a directory 2-146, 2-149
close a stream 2-278
close all files 2-200
close an attribute file 2-73
close system call 2-101
closedir subroutine 2-146, 2-149
closelog subroutine 2-807
closepl subroutine 2-549
closing a DOS file 2-163
clrtoeb subroutine 2-131, 2-246
clrtoel subroutine 2-131, 2-246
cnt_t data type 4-78
code page 4-5, 4-6, 4-7, 4-8, 4-9
 P0 4-6, 4-27
 P1 4-7, 4-34
 P2 4-8, 4-41
 switching 4-9
code point 4-5
 character collation 2-520
code service state 3-203
collector, AIX errors 2-224
color palette, setting 5-87
colorend subroutine 2-246
colorout subroutine 2-247
COLUMNS variable 2-815
command execution
 remote host 2-588, 2-659
communication end point
 See socket
communication, interprocess 2-5, 2-328
communications 5-26
compile regular expression 2-604
complementary error function 2-223
config device driver 5-21
config device driver structure 5-23
config disk structure 5-22
configuration file
 sendmail 3-193
configuration file, security 3-35
configuration file, token-ring 3-242
configuration information, apply 2-69
connect socket subroutine 2-103
connect.com file 3-37
connection
 socket 2-103
construct a unique file name 2-453
construct the name for a temporary file 2-820
cont subroutine 2-549
contents
 directory 1-26
control
 execution of another process 2-567
 file 2-280
 I/O devices 2-407
control characters 4-11
control operations
 shared memory 2-738
control sequence, virtual terminal data 5-78
control sequences 4-13
controlling terminal interface 5-153
controls 4-10
conversion subroutines 2-514
conversion, byte order
 host to network 2-399
 network to host 2-399
convert
 ASCII string to floating-point number 2-798
 string to integer 2-800
convert base-64 ASCII to long integer 2-13
convert between 3-byte integers and long integers 2-416
convert date and time to string 2-120
convert floating-point number to string 2-219
convert formatted input 2-694

convert long integer to base-64 ASCII string 2-13
 converting NLchar string 2-883, 2-885
 copy
 sign of a number 2-109
 copysign subroutine 2-109
 core file 3-43
 cos subroutine 2-764
 cosh subroutine 2-766
 cosine function 2-764
 cpio file 3-45
 cpio structure 3-45
 CPU time used report 2-100
 CRDLY 5-137
 CREAD 5-139
 creat system call 2-110
 create
 interprocess channel 2-545
 new file 2-110
 new process 2-293
 pair of connected sockets 2-772
 socket 2-769
 create a directory 2-178
 create a DOS temporary file 2-180
 create a temporary file 2-819
 create_ipc_prof subroutine 2-113
 creating a DOS file 2-164
 creating backends B-1
 cresetty subroutine 2-247
 crmode subroutine 2-247
 crypt subroutine 2-116
 CR0 5-137
 CR1 5-137
 CR2 5-137
 CR3 5-137
 csavetty subroutine 2-247
 CSIZE 5-139
 CSTOPB 5-139
 ctermid subroutine 2-118
 ctime subroutine 2-120
 ctype macros 2-123
 ctype subroutines 2-123
 current directory
 get path name of 2-396
 current directory, full path name 2-184
 current directory, get path name of 2-345

current directory, path name 2-184
 current DOS Services directory, change 2-160
 current DOS Services drive, change 2-160
 current signal mask
 setting 2-757
 curses subroutine library 2-129
 cursor representation 5-88
 cuserid subroutine 2-140
 customize helper 2-62

D

daddr_t data type 4-78
 daemon, trace 2-824
 DARPA 2-779
 data
 append to a file 2-876
 lock 2-547
 unlock 2-547
 data access
 machine-independent 2-733
 data base subroutines
 data base, terminal capability 3-219
 data blocks 1-24
 data segment 1-8
 change space allocation 2-48
 data stream
 3270 5-26
 data structures
 file system 1-29
 I/O 1-34
 data transport
 See Remote Procedure Calls (RPC)
 data types, defined 4-78
 data types, major
 monitor mode 5-90
 datagrams 2-769
 date format 2-530
 date to string conversion 2-120
 date, modification, change 2-196
 daylight external variable 2-120
 dbm subroutines 2-141
 dbminit subroutine 2-141
 ddi 3-65, 3-165

- ddi file 3-47
- DDS 2-62
- declarations, parameter ix
- Defense Advanced Research Projects Agency 2-779
- Defense Communications Agency 2-779
- define
 - log priority mask 2-807
- Define_Code SVC 5-22
- define_device structure 2-62
- define_device SVC 2-62
- del_ipc_prof subroutine 2-143
- delay_output subroutine 2-131, 2-135
- delch subroutine 2-131, 2-248
- delete a device 2-81
- delete a DOS file 2-198
- delete a minidisk 2-84
- delete protocol procedure 2-869
- delete stanza 2-75
- deleteln subroutine 2-131, 2-248
- delwin subroutine 2-131, 2-248
- describing format of directory entries 4-24
- description file, port 3-173
- description file, security 3-207
- description, file system 3-74
- descriptions file format 3-65
- descriptor
 - file 2-703
- detach
 - shared memory segment 2-741
- determine forms, yes/no response 2-536
- dev_t data type 4-78
- device characteristics 3-66
- device-dependent information 3-65, 3-165
- device driver
 - kernel 1-32
 - VRM 1-32
- device driver, kernel 1-35
- device drivers
 - See also special files
 - definition 1-36
 - trace 5-150
- device I/O 1-36
- device management 1-35
- device number
 - major 1-35
 - minor 1-35
- device status, DOS Services 2-202
- device switch table 1-32
- device, add 2-64
- device, delete 2-81
- devices
 - See special files
- devinfo structure 3-66, 5-119
- DFT 5-26
- dft device driver 5-26
- diacritic characters 4-10
- dir file 3-69
- directory
 - change current 2-87
 - change the root 2-97
 - close 2-146, 2-149
 - create 2-450
 - get path name of current 2-396
 - open 2-146, 2-149
 - read next entry 2-146, 2-149
 - scan 2-701
 - set pointer for reading 2-146, 2-149
- directory change, DOS Services 2-160
- directory contents 1-26
- directory creation 2-178
- directory entry 3-69
 - create a new 2-417, 2-803
 - remove 2-843
- directory entry "." 3-69
- directory entry "." 3-69
- directory entry format, file-system
 - independent 4-24
- directory file 1-19
- directory file structure 3-69
- directory format 3-69
- directory pointer
 - current location 2-146, 2-149
 - reset to beginning 2-146, 2-149
- directory removal, DOS Services 2-190
- directory subroutines 2-146, 2-149
- directory, full path name of current 2-184
- directory, path name of current 2-345
- dirent structure 4-24
- disable auditing 2-31, 2-33
- disclaim system call 2-151
- diskette file 5-33

diskette structure 5-33
 display symbols 4-26
 display, changing physical 5-84
 dispsym definition 4-26
 distance function, euclidean 2-400
 Distributed Function Terminal 5-26
 Distributed Services 2-10
 dn_comp subroutine 2-654
 dn_expand subroutine 2-654
 dn_find subroutine 2-654
 dn_skip subroutine 2-654
 domain
 definition 2-776
 DOS Services assign 2-158
 DOS Services directory, change 2-160
 DOS Services drive, change 2-160
 DOS Services environment, initialize 2-173
 DOS Services file handle duplication 2-166
 DOS Services program execution 2-167
 DOS Services subroutine library 2-153
 DOS Servicesdirectory removal 2-190
 DOS file access 2-153
 DOS file creation 2-164
 DOS file lock 2-176
 DOS file mode, change 2-162
 DOS file modification date, change 2-196
 DOS file read 2-186
 DOS file read/write pointer, move 2-192
 DOS file rename 2-188
 DOS file status, get 2-194
 DOS file system C-1
 DOS file write 2-204
 DOS file, close 2-163
 DOS file, delete 2-198
 DOS file, open 2-182
 DOS file, unlink 2-198
 DOS files synchronization 2-171
 DOS function call table C-2
 DOS function calls C-2
 DOS programs, porting C-1
 DOS temporary file creation 2-180
 dosassign subroutine 2-158
 doschdir subroutine 2-160
 doschmod subroutine 2-162
 dosclose subroutine 2-163
 doscreate subroutine 2-164
 dosdup subroutine 2-166
 dosexecve subroutine 2-167
 dosfirst subroutine 2-169
 dosfstat subroutine 2-194
 dosfsync subroutine 2-171
 dosinit subroutine 2-173
 doslock subroutine 2-176
 dosmkdir subroutine 2-178
 dosmktemp subroutine 2-180
 dosnext subroutine 2-169
 dosopen subroutine 2-182
 dospwd subroutine 2-184
 dosread subroutine 2-186
 dosrename subroutine 2-188
 dosreopen subroutine 2-200
 dosrmdir subroutine 2-190
 dosseek subroutine 2-192
 dosstat subroutine 2-194
 dostouch subroutine 2-196
 dosunlink subroutine 2-198
 dosunopen subroutine 2-200
 dosustat subroutine 2-202
 doswrite subroutine 2-204
 dot notation, Internet 2-402
 double acute accent character 4-10
 dounctrl subroutine 2-249
 doupdate subroutine 2-131
 drand48 subroutine 2-206
 drawbox subroutine 2-249
 drive change, DOS Services 2-160
 drive, DOS Services, assign 2-158
 driver format, message 5-124
 driver, event-tracing 5-150
 drivers
 hft 5-39
 drivers, device
 See special files
 drsname subroutine 2-209
 drsnidd subroutine 2-209
 dsstate system call. 2-212
 dup system call 2-215
 duplicate an open file descriptor 2-215
 duplicating a DOS Services file handle 2-166
 dup2 subroutine 2-217

E

EBCDIC character set 4-46
ecactp subroutine 2-249
ecadpn subroutine 2-249
ecaspn subroutine 2-250
ecblks subroutine 2-250
ecbpls subroutine 2-250
ecbpns subroutine 2-251
ecdflpl subroutine 2-252
ecdppn subroutine 2-252
ecdspl subroutine 2-253
ecdvppl subroutine 2-253
ecflin subroutine 2-254
ECHO 5-140
echo subroutine 2-131, 2-259
ECHOE 5-140
ECHOK 5-140
ECHONL 5-140
ecpnin subroutine 2-259
ecrfpl subroutine 2-260
ecrfpn subroutine 2-260
ecrlpl subroutine 2-261
ecrmppl subroutine 2-261
ecscpn subroutine 2-261
ecshpl subroutine 2-261
ectitl subroutine 2-262
ecvt subroutine 2-219
edata 2-221
effective user ID 2-729
emulation, hft 5-95
enable auditing 2-31, 2-33
encrypt subroutine 2-116
encrypted password 3-168
encryption, password 2-116
end 2-221
endgrent subroutine 2-349
endhostent subroutine 2-353
endnetent subroutine 2-364
endprotoent subroutine 2-374
endpwent subroutine 2-376
endservent subroutine 2-381
endutent subroutine 2-393
endwin subroutine 2-131, 2-262
enhanced signal facilities 2-761

entries in name list, obtaining 2-524
entry
 in system log 2-807
environ global variable 2-227
environment 2-227
environment alteration 2-575
environment facility 4-48
environment setting 3-183
environment subroutines 2-347, 2-520
 getenv 2-347
 NLgetenv 2-347
environment variable, value of 2-347
environment variables
 HOME 4-48
 LANG 4-48
 NLCTAB 4-48
 NLCURSYM 4-48
 NLDATE 4-48
 NLFILE 4-48
 NLLANG 4-48
 NLLDATE 4-48
 NLLDAY 4-48
 NLLMONTH 4-48
 NLNUMSEP 4-48
 NLSDAY 4-48
 NLSMONTH 4-48
 NLSPATH 4-48
 NLTIME 4-48
 NLTMISC 4-48
 NLTSTRS 4-48
 NLTUNITS 4-48
 PATH 4-48
 TERM 4-48
 TZ 4-48
environment, initialize DOS Services 2-173
envp parameter 2-227
eof character 5-134
eol character 5-134
eqn special character definitions 4-56
eqnchar facility 4-56
erand48 subroutine 2-206
erase
 portion of a file 2-276
erase character 5-134
erase subroutine 2-131, 2-262, 2-549
erasechar subroutine 2-131

erf subroutine 2-223
 erfc subroutine 2-223
 errfile file 3-71
 errno 2-543
 errno values A-1
 errno.h A-1
 error codes A-1
 error collector, AIX 2-224
 error file 5-31
 error function 2-223
 error-handling function 2-438
 error logging 5-31
 error messages 2-543
 error numbers A-1
 error values A-1
 errunix subroutine 2-224
 escape sequences 4-13
 etext 2-221
 euclidean distance function 2-400
 event log file 3-71
 event logging 5-31
 event-tracing driver 5-150
 events file format 3-73
 exception handling, floating-point 2-313
 exclusive access
 to a file region 2-427
 exec system call 2-226
 execl system call 2-226
 execle system call 2-226
 execlp system call 2-226
 execute
 file 2-226
 execute a program with a DOS path
 name 2-167
 execution monitor 2-456
 execution profile 2-456
 execution suspension 2-767
 execution time
 profile 2-560
 execv system call 2-226
 execve system call 2-226
 execvp system call 2-226
 exit system call 2-233
 _exit system call 2-233
 exp subroutine 2-235
 exponential function 2-235
 exponentiation 2-235
 expression, regular 2-604, 2-610
 extended AIX system name 2-839
 extended curses subroutine library 2-238
 extended curses 2-238
 extended message receive 2-493
 extended read 2-591
 extended subroutine 2-262
 eXternal Data Representation (XDR)
 See also Remote Procedure Calls (RPC)
 bit fields 2-888
 bit maps 2-888
 data blocks 2-888
 data representation
 arrays, counted 2-891
 arrays, fixed 2-891
 arrays, fixed-length 2-900
 basic data types 2-893
 basic data types, overview 2-894
 Booleans 2-889
 constructed data types 2-893
 counted byte strings 2-891
 data stream access routines,
 overview 2-904
 differences from C constructs 2-888
 discriminated unions 2-892
 double precision 2-890
 enumerations 2-889
 floating-point 2-889
 integers 2-888
 opaque data 2-890
 structures 2-892
 unsigned integers 2-889
 data translation 2-887
 defined 2-887
 library routines
 basic data types, overview 2-894
 filter primitives, overview 2-893
 filters, constructed data type 2-897
 filters, floating-point numbers 2-896
 filters, generic enumeration 2-896
 filters, double precision number 2-896
 non-filter primitives 2-903
 overview 2-893
 primitives, basic data 2-893
 primitives, constructed data 2-893

- record stream utilities, overview 2-906
- return values 2-893
- with C programs 2-893
- xdr_array subroutine 2-899
- xdr_bool subroutine 2-896
- xdr_bytes subroutine 2-898
- xdr_double precision number
subroutine 2-896
- xdr_enum subroutine 2-896
- xdr_floats subroutine 2-896
- xdr_getpos subroutine 2-903
- xdr_opaque subroutine 2-902
- xdr_reference subroutine 2-903
- xdr_string subroutine 2-897
- xdr_union subroutine 2-902
- xdr_void subroutine 2-897
- xdrmem_create subroutine 2-905
- xdrrec_create subroutine 2-906
- xdrrec_endofrecord subroutine 2-907
- xdrrec_eof subroutine 2-907
- xdrrec_skiprecord subroutine 2-907
- xdrs parameter 2-893
- xdrstdio_create subroutine 2-905
- relationship to C constructs 2-887
- standard types 2-887
- syntax 2-887
- externals
 - edata 2-221
 - end 2-221
 - etext 2-221

F

- F_DUPFD 2-280
- F_GETFD 2-280
- F_GETFL 2-281
- F_GETLK 2-281
- F_SETFD 2-281
- F_SETFL 2-281
- F_SETLK 2-282
- F_SETLKW 2-282
- fabs subroutine 2-290
- facilities
 - mm 4-66

- regex 2-610
- facilities, miscellaneous
 - See miscellaneous facilities
- fault generation, IOT 2-15
- fchmod system call 2-89
- fchown system call 2-93
- fclear system call 2-276
- fclose subroutine 2-278
- fcntl system call 2-280
- fcntl.h header file 4-58
- fcvt subroutine 2-219
- fd devinfo structure 5-34
- fd file 5-33
- fdopen subroutine 2-291
- feof macro 2-285
- ferror macro 2-285
- fetch subroutine 2-141
- FFDLY 5-137
- fflush subroutine 2-278
- ffs subroutine 2-52
- ffullstat system call 2-334
- FF0 5-137
- FF1 5-138
- fgetc subroutine 2-342
- fgets subroutine 2-379
- fgetwc subroutine 2-342
- fgetws subroutine 2-379
- fifo
 - create 2-450
- file 2-538, 2-591
 - accessibility, determine 2-19
 - close a 2-101
 - control 2-280
 - create 2-450
 - creation 2-110
 - directory entry
 - create a new 2-417, 2-803
 - erase portion of 2-276
 - execute 2-226
 - lock a region 2-427
 - mode change 2-89
 - open to read or write 2-538
 - read from 2-591
 - read from, extended 2-591
 - rewrite 2-110
 - shorten 2-330

- unlock a region 2-427
- write 2-876, 2-880
- write changes 2-326
- file access
 - set time 2-850
- file control 2-3
- file creation mask
 - get 2-836
 - set 2-836
- file creation, DOS, temporary 2-180
- file creation, temporary 2-819
- file descriptor 2-703
 - close 2-101
 - duplication 2-215, 2-217
 - get table size 2-346
- file entry, group, obtaining 2-349
- file entry, utmp access 2-393
- file formats
 - archive 3-18
 - audit 3-22
 - audit trail 3-22
 - events 3-73
 - process accounting 3-15
 - sendmail.cf 3-193
 - serverattach 3-203
- file i/o subsystem 1-32
- file locks 2-281
 - read lock 2-281
 - write lock 2-281
- file maintenance 2-3
- file mapping 2-8
- file member, archive structure 3-18
- file mode change, DOS 2-162
- file modification
 - set time 2-850
- file name generation, terminal 2-118
- file name, construct 2-453
- file name, make 2-453
- file naming, temporary files 2-820
- file pointer
 - read/write 2-434
- file pointer repositioning 2-324
- file status
 - obtain 2-786
- file status, DOS, get 2-194
- file synchronization, DOS 2-171
- file system
 - backup format 3-24
 - data structures 1-29
 - DOS C-1
 - layout 1-21
 - mount 2-460
 - statistics 2-848
 - unmount 2-837
- file system attributes 3-74
- file system description 3-74
- file system management 1-18
- file system status
 - obtain 2-790
- file system table 3-161
- file time, set 2-853
- file tree, read 2-332
- file types 1-19
 - directory 1-19
 - ordinary 1-20
 - special 1-20
- file, assembler output 3-5
- file, link editor output 3-5
- file, storage image 3-43
- fileno macro 2-285
- files
 - directory 2-450
 - header x
 - mapped 2-8
 - ordinary 2-450
 - special 1-36, 2-450, 2-460
- files, device
 - See special files
- files, special
 - See special files
- filesystems file 3-74
- find DOS files that match a pattern 2-169
- find slot in utmp file for current user 2-833
- find value of user information name 2-392
- find_ipc_prof subroutine 2-287
- firstkey subroutine 2-141
- fixterm subroutine 2-131
- flag letter, get from argument vector 2-367
- flash subroutine 2-131, 2-262
- floating-point
 - conversion from ASCII 2-798
- Floating-Point Accelerator 2-295, 2-308

floating-point exception handling 2-313
 floating-point numbers manipulation 2-322
 floating point operations
 copy sign 2-109
 floating-point subroutines, ANSI/IEEE 2-295
 floating-point to string conversion 2-219
 floor function 2-290
 floor subroutine 2-290
 flush a stream 2-278
 flushinp subroutine 2-131
 fmod subroutine 2-290
 font file format 3-79
 font symbols 4-26
 fonts, changing 5-87
 fopen subroutine 2-291
 fork 2-293
 form ix
 format ix, 2-530
 date 2-530
 time 2-530
 format file, message system 3-141
 format of cpio archive 3-45
 format specification, text files 3-108
 format, archive 3-18
 format, gps 3-110
 format, message driver 5-124
 format, system volume 3-100
 formats
 directory 3-69
 directory entry, file-system
 independent 4-24
 event log file 3-71
 inode 3-119
 master 3-125
 formats, file
 See file formats
 formatted input conversion 2-694
 formatted output, print 2-553
 formatted varargs argument list, print 2-866
 formatting a permuted index, macro
 package 4-67
 forward file 3-151
 FP_DOUBLE 2-295
 FP_FLOAT 2-295
 fpfp subroutines 2-295
 fprintf subroutine 2-553
 fputc subroutine 2-572
 fputs subroutine 2-581
 fputwc subroutine 2-572
 fputws subroutine 2-581
 frame
 token-ring
 format checking 2-665
 fread subroutine 2-320
 free-block list 1-24
 free blocks
 allocation 1-25
 free subroutine 2-436
 freopen subroutine 2-291
 frexp subroutine 2-322
 fs file 3-100
 fscanf subroutine 2-694
 fseek subroutine 2-324
 fspec file 3-108
 fstat system call 2-786
 fstatfs system call 2-790
 fsync system call 2-326
 ftell subroutine 2-324
 ftok subroutine 2-328
 ftruncate system call 2-330
 ftw subroutine 2-332
 fullbox subroutine 2-263
 fullstat structure 4-60
 fullstat system call 2-334
 fullstat.h header file 4-60
 function calls
 DOS C-2
 function libraries
 See libraries
 function, complementary error 2-223
 function, error 2-223
 function, error-handling 2-438
 function, euclidean distance 2-400
 functions
 See also kernel subroutines
 See also system calls and subroutines
 absolute value 2-290
 ceiling 2-290
 floor 2-290
 remainder 2-290
 functions hyperbolic 2-766
 functions, trigonometric 2-764

fwrite subroutine 2-320

G

gamma function 2-340
gamma subroutine 2-340
gcvf subroutine 2-219
generate file name for terminal 2-118
generate pseudo-random numbers 2-585, 2-586
generating an IOT fault 2-15
geometric text font 3-88
get
 file status 2-786
 file system status 2-790
 group IDs 2-391
 message queue identifier 2-468
 process IDs 2-373
 time 2-816
 user IDs 2-391
get a string from a stream 2-379
get character or word from stream 2-342
get DOSfile status 2-194
get file system statistics 2-848
get group file entry 2-349
get login name 2-362
get names from name list 2-524
get option letter from argument vector 2-367
get password file entry 2-376
get path name of current directory 2-345
get status of DOS Services device 2-202
get the name of a terminal 2-832
get user name 2-140
getc macro 2-342
getch subroutine 2-131, 2-263
getchar macro 2-342
getcwd subroutine 2-345
getdtablesize subroutine 2-346
getegid system call 2-391
getenv subroutine 2-347
geteuid system call 2-391
getgid system call 2-391
getgrent subroutine 2-349
getgrgid subroutine 2-349
getgrnam subroutine 2-349

getgroups system call 2-351
gethostbyaddr subroutine 2-353
gethostbyname subroutine 2-353
gethostid socket subroutine 2-356
gethostname socket subroutine 2-358
getitimer subroutine 2-360
getlogin subroutine 2-362
getlong subroutine 2-654
getnetbyaddr subroutine 2-364
getnetbyname subroutine 2-364
getnetent subroutine 2-364
getopt subroutine 2-367
getpass subroutine 2-370
getpeername socket subroutine 2-371
getpgrp system call 2-373
getpid system call 2-373
getppid system call 2-373
getprotobyname subroutine 2-374
getprotobynumber subroutine 2-374
getprotoent subroutine 2-374
getpwent subroutine 2-376
getpwnam subroutine 2-376
getpwuid subroutine 2-376
gets subroutine 2-379
getservbyname subroutine 2-381
getservbyport subroutine 2-381
getservent subroutine 2-381
getshort subroutine 2-654
getsockname socket subroutine 2-383
getsockopt socket subroutine 2-385
getstr subroutine 2-131, 2-264
gettimeofday 2-389
getmode subroutine 2-131, 2-264
getuid system call 2-391
getuinfo subroutine 2-392
getutent subroutine 2-393
getutid subroutine 2-393
getutline subroutine 2-393
getw subroutine 2-342
getwc macro 2-342
getwc subroutine 2-342
getwchar macro 2-342
getwchar subroutine 2-342
getwd subroutine 2-396
getws subroutine 2-379
getyx subroutine 2-131, 2-264

gmtime subroutine	2-120	gsfci subroutine	6-76
goto, nonlocal	2-726	gsfell subroutine	6-78
gps format	3-110	gsfply subroutine	6-80
graphic output file format	3-171	gsfrec subroutine	6-82
graphic symbols	4-26	gsgtat subroutine	6-84
graphics interface	3-171	gsgtxt subroutine	6-89
graphics interface subroutines		gsignal subroutine	2-781
grave accent character	4-10	gsinit subroutine	6-91
Greek characters	4-62	GSL	6-2
greek facility	4-62	gslatt subroutine	6-95
group access list	2-404	gslcat subroutine	6-97
get	2-351	gsline subroutine	6-99
set	2-724	gslock subroutine	6-101
group file	3-113	gslop subroutine	6-103
group file entry, obtaining	2-349	gslpat subroutine	6-106
group file entry, write	2-577	gsmask subroutine	6-108
group ID		gsmatt subroutine	6-110
set	2-731	gsmcat subroutine	6-113
set for a process	2-728	gsmcur subroutine	6-116
group ID of a file		gsmfld subroutine	6-118
change	2-93	gsmult subroutine	6-123
group ID translation	2-93	gspcls subroutine	6-125
group IDs		gsplym subroutine	6-127
get	2-391	gspoly subroutine	6-129
gsbply subroutine	6-28	gspp subroutine	6-131
gscarc subroutine	6-30	gsqdsp subroutine	6-133
gscatt subroutine	6-32	gsqfnt subroutine	6-136
gscenv subroutine	6-35	gsqgtx subroutine	6-138
gscir subroutine	6-38	gsqlex subroutine	6-140
gsclrs subroutine	6-40	gsqloc subroutine	6-142
gscmap subroutine	6-41	gsrrst subroutine	6-144
gscrca subroutine	6-43	gsrsav subroutine	6-146
gsdply subroutine	6-45	gssend subroutine	6-149
gsdpik subroutine	6-47	gstatt subroutine	6-153
gsdara subroutine	6-49	gsterm subroutine	6-156
gsdarc subroutine	6-51	gstext subroutine	6-157
gsecnv subroutine	6-53	gsulns subroutine	6-159
gsecur subroutine	6-56	gsunlk subroutine	6-161
gsell subroutine	6-57	gsvgrn subroutine	6-162
gsepik subroutine	6-59	gsxblt subroutine	6-164
gseply subroutine	6-61	gsxcnv subroutine	6-171
gsevds subroutine	6-63	gsxpvr subroutine	6-173
gseven subroutine	6-65	gsxtat subroutine	6-175
gsevt subroutine	6-67	gsxtxt subroutine	6-180
gsfatt subroutine	6-74		

H

handle, duplicating 2-166
hardware access
 RT C-7
has_ic subroutine 2-131
has_il subroutine 2-131
hash tables 2-397
hcreate subroutine 2-397
HD devinfo structure 5-37
hdestroy subroutine 2-397
head, of screen manager ring 5-70
header files x
help text, issue 2-471
help text, retrieve 2-486
helper, customize 2-62
hft device, query 2-814
hft driver 5-39
hft emulation 5-95
hft, remote 5-95
history file 3-116
hole
 make in a file 2-276
HOME environment variable 4-48
host byte order
 conversion to network byte order 2-399
host identifier 2-356
host name 2-358
hostent structure 2-354
hsearch subroutine 2-397
htonl subroutine 2-399
htons subroutine 2-399
HUPCL 5-139
hyperbolic cosine function 2-766
hyperbolic functions 2-766
hyperbolic sine function 2-766
hyperbolic tangent function 2-766
hypot subroutine 2-400

I

i-list layout 1-22
i-node layout 1-23
i-nodes
 update 2-806
i-number allocation 1-24
I/O 2-3
I/O activity
 wait for 2-703
I/O bus 1-33
I/O data structures 1-34
I/O devices
 See also special files
 control operations 2-407
I/O overview 1-30
I/O status
 check 2-703
I/O, buffered 2-783
ICANON 5-140
ICRNL 5-136
idlok subroutine 2-131
IEEE floating point 2-295
ieeetrap subroutine 2-314
IGNBRK 5-136
IGNCR 5-136
IGNPAR 5-136
ilog file 3-246
image, memory 5-122
image, virtual memory 5-122
immediate message, issue 2-474
inch subroutine 2-132, 2-264
index subroutine 2-401
inet_addr subroutine 2-402
inet_lnaof subroutine 2-402
inet_makeaddr subroutine 2-402
inet_netof subroutine 2-402
inet_network subroutine 2-402
inet_ntoa subroutine 2-402
information
 handling name server 2-654
.init.state file format 3-3
initgroups subroutine 2-404
initial AIX state 3-3
initialize DOS Services environment 2-173

initialize group access list 2-404
 initiate a pipe to or from a process 2-551
 initscr subroutine 2-132, 2-265
 initstate subroutine 2-586
 INLCR 5-136
 ino_t data type 4-78
 inode format 3-119
 inode structure 3-119
 INPCK 5-136
 input
 read into multiple buffers 2-597
 input stream, put character back 2-841
 input/output 2-3
 input/output devices
 control operations 2-407
 input/output, buffered 2-783
 input/output, device 1-36
 input, binary 2-320
 inquiry, stream status 2-285
 insch subroutine 2-132, 2-265
 insert
 queue element 2-406
 insert mode 5-86
 insert, retrieve 2-486
 insertln subroutine 2-132, 2-265
 insque subroutine 2-406
 install protocol procedure 2-869
 integer
 conversion from string 2-800
 integer absolute value 2-16
 integer to ASCII conversion 2-13
 interface control, terminal 5-153
 interface, graphics 3-171
 internal data structures
 dirent 4-24
 internal timer
 get value of 2-360
 set value of 2-360
 international character support 2-530
 character classification 2-500
 character collation 2-497
 character conversion 2-105
 date format 2-530
 environment 2-520, 4-48
 formatted output 2-553
 NLchar data type 2-514
 parameter fetching 2-522
 string conversion 2-516
 string handling 2-530, 2-533
 string operations 2-503, 2-526
 time format 2-530
 time structure 2-533
 Internet address
 manipulation 2-402
 Internet dot notation 2-402
 interprocess channel
 create 2-545
 interprocess communication 2-5, 2-328
 intr character 5-134
 intrflush subroutine 2-132
 ioctl system call 2-407
 IOT fault generation 2-15
 IPC 2-5
 ipc_perm structure 2-7
 IPC_RMID 2-739
 IPC_SET 2-739
 IPC_STAT 2-738
 IPL 3-3
 virtual machine 2-409, 2-599
 iplvm system call 2-409
 isalnum macro 2-123
 isalpha macro 2-123
 isascii macro 2-123
 isatty subroutine 2-832
 isctrl macro 2-123
 isdigit macro 2-123
 isgraph macro 2-123
 ISIG 5-140
 isjalpha macro 2-123
 isjdigit macro 2-123
 isjgraph macro 2-123
 isjhira macro 2-123
 isjis macro 2-123
 isjkanji macro 2-123
 isjkata macro 2-123
 isjlower macro 2-123
 isjparen macro 2-123
 isjprint macro 2-123
 isjpunct macro 2-123
 isjspace macro 2-123
 isjupper macro 2-123
 isjxdigit macro 2-123

- islower macro 2-123
- isprint macro 2-123
- ispunct macro 2-123
- isspace macro 2-123
- issue a queued message 2-478
- issue a shell command 2-810
- issue an immediate message 2-474
- issue help text 2-471
- ISTRIP 5-136
- isupper macro 2-123
- isxdigit macro 2-123
- IUCLC 5-136
- IXANY 5-136
- IXOFF 5-136
- IXON 5-136

J

Japanese Language Support

- catclose 2-54
- catgetmsg 2-56
- catgets 2-58
- catopen 2-60
- NLcatgets 2-510
- NLcatopen 2-512
- NLgetamsg 2-518
- jistoa subroutine 2-105
- jistoa subroutine 2-105
- jrand48 subroutine 2-206
- j0, j1, jn subroutines 2-44

K

- kaf file format 3-121
- kernel calls
 - See kernel subroutines
 - See system calls and subroutines
- kernel device driver 1-32
- kernel features 1-4
- kernel functions
 - file system management 1-3
 - input/output control 1-4

- memory management 1-4
- process management 1-3
- program management 1-4
- resource management 1-4
- time management 1-4
- kernel mode 1-6
- kernel mode addressing 1-10
- kernel overview 1-3
- kernel subroutines
- kernel trap routine 1-31
- kernel, rebuild 2-69
- key_t data type 4-78
- keyboard 5-98
- keypad subroutine 2-132, 2-266
- keywords, ddi 3-65
- kill character 5-134
- kill system call 2-411
- killchar subroutine 2-132
- killpg subroutine 2-413
- kmem file 5-122
- kuentojis subroutine 2-105

L

- label subroutine 2-549
- label_t data type 4-78
- LANG environment variable 2-60, 2-512, 4-48
- layout
 - block 0 1-21
 - file system 1-21
 - i-list 1-22
 - i-node 1-23
 - superblock 1-21
- lcong48 subroutine 2-206
- ldexp subroutine 2-322
- leaveok subroutine 2-132, 2-266
- letter, option, get from argument vector 2-367
- level_t data type 4-78
- lfind subroutine 2-432
- libPW subroutine library 2-563
- libraries
 - DOS Services 2-153
 - programmers workbench 2-563
 - Ring Error Monitor 2-661

- sockets 2-774
- standard I/O 2-783
- libsock subroutine library 2-774
- light-emitting diodes, setting 5-81
- limits
 - user 2-834
- line buffer mode 2-722
- line subroutine 2-549
- linear congruential algorithm 2-206
- linear search and update 2-432
- linemod subroutine 2-549
- LINES variable 2-815
- link
 - create 2-417, 2-803
- link editor output file 3-5
- link system call 2-417
- list
 - free-block 1-24
- listen
 - for socket connection 2-420
- listen subroutine 2-420
- loadtbl system call 2-422
- localtime subroutine 2-120
- locator thresholds 5-81
- lock
 - data 2-547
 - process 2-547
 - region of a file 2-427
 - text 2-547
- lock a region of a DOS file 2-176
- lockf system call 2-427
- log file
 - close 2-807
 - define priority mask 2-807
 - open 2-807
- log file, token-ring 3-244
- log priority mask 2-807
- log subroutine 2-235
- logarithm 2-235
- login name 2-140
- login name of user, obtaining 2-431
- login name, get 2-362
- login, remote 5-126
- logname subroutine 2-431
- log10 subroutine 2-235
- long integers from 3-byte integers 2-416

- longjmp subroutine 2-726
- longname subroutine 2-132, 2-266
- lp special file 5-117
- lprio structure 5-119
- lprmode structure 5-119
- LPRUDE structure 5-120
- lrnd48 subroutine 2-206
- lsearch subroutine 2-432
- lseek system call 2-434
- lstat system call 2-786
- ltol3 subroutine 2-416
- l3tol subroutine 2-416
- l64a subroutine 2-13

M

- machine-independent data access 2-733
- macro definitions x
- macro package for formatting a permuted index 4-67
- macron accent character 4-10
- macros
 - _atojis 2-105
 - _jistoa 2-105
 - _NCtolower 2-105
 - _NCtoupper 2-105
 - _tojlower 2-105
 - _tojupper 2-105
 - _tolower 2-105
 - _toupper 2-105
- clearerr 2-285
- ctype 2-123
- feof 2-285
- ferror 2-285
- fileno 2-285
- getc 2-342
- getchar 2-342
- getwc 2-342
- getwchar 2-342
- isalnum 2-123
- isalpha 2-123
- isascii 2-123
- isctrl 2-123
- isdigit 2-123

- isgraph 2-123
- isjalpha 2-123
- isjdigit 2-123
- isjgraph 2-123
- isjhira 2-123
- isjis 2-123
- isjkanji 2-123
- isjkata 2-123
- isjlower 2-123
- isjparen 2-123
- isjprint 2-123
- isjpunct 2-123
- isjspace 2-123
- isjupper 2-123
- isjxdigit 2-123
- islower 2-123
- isprint 2-123
- ispunct 2-123
- isspace 2-123
- isupper 2-123
- isxdigit 2-123
- NLyesno 2-536
- putc 2-572
- putchar 2-572
- putwc 2-572
- putwchar 2-572
- varargs 2-858
- magic number 2-226
- maildelivery file 3-151
- main memory allocator 2-436
- main subroutine 2-227
- maintenance 2-3
- maintenance mode 3-3
- make
 - hole in a file 2-276
- make a unique file name 2-453
- malloc subroutine 2-436
- management
 - device 1-35
- manipulate parts of floating-point
 - numbers 2-322
- manipulating
 - Internet addresses 2-402
- mapped file
 - attach 2-734
- mapped files 2-8
- mask
 - file creation 2-836
- master file 3-125
- master format 3-125
- match regular expression 2-604
- math.h header file 4-64
- matherr subroutine 2-438
- mdverify subroutine 2-443
- medium access control frames 3-136
- mem file 5-122
- memccpy 2-445
- memchr 2-445
- memcmp 2-445
- memcpy 2-445
- memory allocator 2-436
- memory control operations
 - shared 2-738
- memory image 5-122
- memory image file 5-122
- memory management 1-6
- memory-mapped files 2-8
- memory operations 2-445
- memory segment
 - attach to process 2-734
 - detach 2-741
 - get 2-743
- memory subroutine 2-445
- memory, disclaim 2-151
- memset 2-445
- message
 - control operations 2-463
 - from queue 2-482
 - receive from a socket 2-601
 - send to a socket 2-718
- message control 2-463
- message driver format 5-124
- message file 3-133
- message protocol
 - See Remote Procedure Calls (RPC)
- message queue 2-703
 - get identifier 2-468
 - send message 2-490
- message receive
 - extended 2-493
- message system alias file 3-138
- message system format file 3-141

message, issue a queued 2-478
 message, issue an immediate 2-474
 message, retrieve 2-486
 messages, error 2-543
 meta subroutine 2-132, 2-266
 Mfile 3-136
 mh-alias file 3-138
 mh-format file 3-141
 mh-mail file 3-148
 mh-profile file 3-155
 mh-tailor file 3-159
 mhook 3-151
 minidisk customizing 5-36
 minidisk, add 2-67
 minidisk, delete 2-84
 minidisks 2-443
 miscellaneous facilities
 mkdir system call 2-447
 mknod system call 2-450
 mktemp subroutine 2-453
 mm facility 4-66
 mm macro package 4-66
 mntctl system call 2-454
 mnttab file 3-161
 mnttab.h structure 3-161
 mode bit
 set-group-ID 2-228
 set-user-ID 2-228
 mode change, file 2-89
 mode, DOS file, change 2-162
 modes
 kernel 1-6
 user 1-6
 modf subroutine 2-322
 modification date, change, DOS file 2-196
 modification time
 file 2-850
 monitor mode major data type 5-90
 monitor subroutine 2-456
 mount
 file system 2-460
 mount system call 2-460
 mounted file system table 3-161
 move
 read/write file pointer 2-434
 move DOS file read/write pointer 2-192
 move subroutine 2-132, 2-267, 2-549
 mptx facility 4-67
 mrand48 subroutine 2-206
 msgbuf structure 2-482
 msgctl system call 2-463
 msgget system call 2-468
 msghdr structure 2-602
 msghelp subroutine 2-471
 msgimed subroutine 2-474
 msgop system calls 2-482, 2-490, 2-493
 msgqued subroutine 2-478
 msgrcv system call 2-482
 msggrtv subroutine 2-486
 msgsnd system call 2-490
 msgxrcv system call 2-493
 mtstailor file 3-159
 multibyte characters 4-9
 multibyte controls 4-13
 multiple buffers
 read input 2-597
 Multiprotocol Adapter 5-26
 multiuser mode 3-3
 mv facility 4-68
 mvaddch subroutine 2-132, 2-242
 mvaddstr subroutine 2-130, 2-243
 mvchgat subroutine 2-244
 mvcur subroutine 2-132, 2-267
 mvdelch subroutine 2-132, 2-248
 mvgetch subroutine 2-132, 2-263
 mvgetstr subroutine 2-132, 2-264
 mvinch subroutine 2-132, 2-264
 mvinsch subroutine 2-132, 2-265
 mvpaddch subroutine 2-242
 mvpaddstr subroutine 2-243
 mvpchgat subroutine 2-244
 mvprintw subroutine 2-132
 mvscanw subroutine 2-132
 mvwaddch subroutine 2-130, 2-242
 mvwaddstr subroutine 2-130, 2-243
 mvwchgat subroutine 2-244
 mvwdelch subroutine 2-132, 2-248
 mvwgetch subroutine 2-132, 2-263
 mvwgetstr subroutine 2-132, 2-264
 mvwin subroutine 2-132, 2-267
 mvwinch subroutine 2-132, 2-264
 mvwinsch subroutine 2-132, 2-265

mvwprintw subroutine 2-133
mvwscanw subroutine 2-133

N

name for a temporary file, create 2-820
name list entries, obtaining 2-524
name of a terminal 2-832
name of the user 2-140
name server
 interpret information 2-654
 resolver subroutines 2-654
 send information 2-654
name, login 2-362
name, user login, obtaining 2-431
name, user, find value 2-392
NCchrlen macro 2-514
NCcollate subroutine 2-497
NCcolumniq subroutine 2-497
NCcolval subroutine 2-497
NCctype 2-500
NCdec macro 2-514
NCdechr macro 2-514
NCdecode subroutine 2-514
NCdecstr subroutine 2-514
NCenc macro 2-514
NCencode subroutine 2-514
NCencstr subroutine 2-514
NCeqvmap subroutine 2-497
NCesc subroutine 2-105
NCflatchar subroutine 2-105
NCisalnum subroutine 2-500
NCisalpha subroutine 2-500
NCiscentrl subroutine 2-500
NCisdigit subroutine 2-500
NCisgraph subroutine 2-500
NCislower subroutine 2-500
NCisNLchar subroutine 2-500
NCisprint subroutine 2-500
NCispunct subroutine 2-500
NCisshift subroutine 2-500
NCisspace subroutine 2-500
NCisupper subroutine 2-500
NCisxdigit subroutine 2-500

NCstrcat subroutine 2-503
NCstrchr subroutine 2-503
NCstrcmp subroutine 2-503
NCstrcpy subroutine 2-503
NCstrcspn subroutine 2-503
NCstring subroutine 2-503
NCstrlen subroutine 2-503
NCstrncat subroutine 2-503
NCstrncmp subroutine 2-503
NCstrncpy subroutine 2-503
NCstrpbrk subroutine 2-503
NCstrrchr subroutine 2-503
NCstrspn subroutine 2-503
NCstrtok subroutine 2-503
_Nctolower macro 2-105
Nctolower subroutine 2-105
NCtoNLchar subroutine 2-105
_Nctoupper macro 2-105
Nctoupper subroutine 2-105
NCunesc subroutine 2-105
NCwunesc subroutine 2-105
_NCxcol macro 2-497
neqn special character definitions 4-56
netent structure 2-364
network byte order
 conversion to host byte order 2-399
network data base
 close 2-364
 find an entry in 2-364
 open 2-364
network data base entry 2-364
Network File System 2-10
network host address 2-353
network host data base
 close 2-353
 find an entry in 2-353
 open 2-353
network host name 2-353
Network Information Center 2-779
network protocol address 2-374
network protocol data base
 close 2-374
 find an entry in 2-374
 open 2-374
network protocol name 2-374
network service address 2-381

network service name	2-381	NLNUMSEP environment variable	4-48
network services data base		NLO	5-137
close	2-381	NLprintf subroutine	2-553, 2-558
find an entry in	2-381	NLscanf subroutine	2-694
open	2-381	NLSDAY environment variable	4-48
new-line character	5-134	NLSMONTH environment variable	4-48
new process image	2-226	NLSPATH environment variable	2-60, 2-512, 4-48
newpad subroutine	2-133	NLsprintf subroutine	2-553
newterm subroutine	2-133	NLsscanf subroutine	2-694
newview subroutine	2-267	NLstrcat subroutine	2-526
newwin subroutine	2-133, 2-268	NLstrchr subroutine	2-526
nextkey subroutine	2-141	NLstrcmp subroutine	2-526
NFS	2-10	NLstrcpy subroutine	2-526
nice system call	2-508	NLstrcspn subroutine	2-526
nl subroutine	2-133, 2-268	NLstring	2-526
NLcatgets subroutine	2-510-2-511	NLstrlen subroutine	2-526
NLcatopen subroutine	2-512-2-513	NLstrncat subroutine	2-526
NLchar data type	2-514	NLstrncmp subroutine	2-526
NLchar string, converting	2-883, 2-885	NLstrncpy subroutine	2-526
data translation		NLstrpbrk subroutine	2-526
serialization	2-887	NLstrrchr subroutine	2-526
relationship to C constructs		NLstrspn subroutine	2-526
passing addresses	2-887	NLstrtime subroutine	2-530
NLchrln macro	2-514	NLstrtok subroutine	2-526
NLconvstr subroutines	2-516	NLTIME environment variable	4-48
NLcplen subroutine	2-526	NLTMISC environment variable	4-48
NLCTAB environment variable	4-48	NLtmtime subroutine	2-533
NLCURSYM environment variable	4-48	NLTSTRS environment variable	4-48
NLDATE environment variable	4-48	NLTUNITS environment variable	4-48
NLDLY	5-137	NLunescstr subroutine	2-516
NLecflin subroutine	2-254	NLvfprintf subroutine	2-866
NLescstr subroutine	2-516	NLvprintf subroutine	2-866
NLFILE environment variable	4-48	NLvsprintf subroutine	2-866
NLflatstr subroutine	2-516	_NLxcol macro	2-497
NLfprintf subroutine	2-553	NLyesno macro	2-536
NLfscanf subroutine	2-694	NLyesno subroutine	2-536
NLgetamsg subroutine	2-518-2-519	NL1	5-137
NLgetctab subroutine	2-520	nocbreak subroutine	2-131
NLgetenv subroutine	2-347	nocrmode subroutine	2-247
NLgetfile	2-522	nodelay subroutine	2-131, 2-269
NLisNLcp macro	2-514	noecho subroutine	2-131, 2-259
nlist subroutine	2-524	NOFLSH	5-140
NLLANG environment variable	4-48	nometa subroutine	2-266
NLLDATE environment variable	4-48	non-standard tabbing	3-108
NLLDAY environment variable	4-48	nonl subroutine	2-133, 2-268
NLLMONTH environment variable	4-48		

nonlocal goto 2-726
nonspacing characters 4-10
noraw subroutine 2-133, 2-270
nrand48 subroutine 2-206
ntohl subroutine 2-399
ntohs subroutine 2-399
null special file 5-123
number
 magic 2-226
numbers, pseudo-random 2-206, 2-585, 2-586
nvram file 5-122

O

OCRNL 5-137
OFDEL 5-137
OFILL 5-137
ogonek accent character 4-10
OLCUC 5-137
ONLCR 5-137
ONLRET 5-137
ONOCR 5-137
open
 directory 2-146, 2-149
 log file 2-807
 network data base 2-364
 network host data base 2-353
 network protocol data base 2-374
 network services data base 2-381
open a DOS file 2-182
open a stream 2-291
open attribute file 2-77
open file
 to read 2-538
 to write 2-538
open system call 2-538
opendir subroutine 2-146, 2-149
openlog subroutine 2-807
openpl subroutine 2-549
operating system profiler 5-125
OPOST 5-137
oprmode structure 5-120
option letter, get from argument vector 2-367
options

socket 2-385
options file format 3-165
ordinary file 1-20
osm driver 5-124
out-of-band data 2-387
output file, assembler 3-5
output file, link editor 3-5
output, binary 2-320
output, print formatted 2-553
overcircle accent character
overdot accent character 4-10
overlay subroutine 2-133, 2-269
overview
 I/O 1-30
 signals 2-5
overview of kernel 1-3
overview of sockets 2-776
overwrite subroutine 2-133, 2-269
owner ID translation 2-93
owner of a file 2-93
 change 2-93

P

paddch subroutine 2-242
paddr_t data type 4-78
paddstr subroutine 2-243
palette, setting color 5-87
param.h header file 4-72
parameter passing 2-227
parameters ix
PARENB 5-139
parent directory 3-69
parent process 1-13, 2-293
parent process ID 2-373
PARMRK 5-136
PARODD 5-139
passing
 parameter 2-227
passwd file 3-167
password description 3-168
password encryption 2-116
password file entry, get 2-376
password file entry, write 2-579

password, read 2-370
 PATH environment variable 4-48
 path name
 absolute 1-26
 relative 1-28
 resolution 1-26
 path name of current directory 2-345
 pattern, finding DOS files that match 2-169
 pause system call 2-542
 PC-DOS programs, porting C-1
 pchgat subroutine 2-244
 pclose subroutine 2-551
 pcs font 3-88
 peer
 definition 2-371
 peer name
 socket 2-371
 pel box
 definition 3-98
 perase subroutine 2-262
 permanent storage
 write file to 2-326
 permission
 file access 2-89
 perror subroutine 2-543
 physadr structure 4-78
 pipe initiation 2-551
 pipe system call 2-545
 pixel map 6-167
 plock system call 2-547
 plot file format 3-171
 plot subroutines 2-549
 pnoutrefresh subroutine 2-133
 point subroutine 2-549
 pointer, DOS file read/write, move 2-192
 pointer, scan directory 2-86
 popen subroutine 2-551
 port description file 3-173
 porting DOS 3.0 C-1
 ports file 3-173
 portstatus file 3-178
 portstatus structure 3-178
 pow subroutine 2-235
 power (exponentiation) 2-235
 predefined file 3-180
 prefresh subroutine 2-133
 prf file 5-125
 print
 formatted output 2-553
 print floating-point number 2-219
 print formatted varargs argument list 2-866
 printf subroutine 2-553
 printw subroutine 2-133, 2-270
 priority computation 1-16
 priority of a process
 change 2-508
 privileged address 2-588
 process
 child 1-13
 creation 2-293
 get audit state 2-41
 get IDs 2-373
 get owner 2-846
 lock 2-547
 parent 1-13
 preemption 1-15
 set audit state 2-41
 set owner 2-846
 states 1-15
 trace execution 2-567
 unlock 2-547
 process accounting 2-22
 process accounting file 3-15
 process addressing 1-6
 process alarm 2-25
 process audit state 2-41
 process communication
 signals 1-16
 process control 1-5, 2-4
 of process execution 2-567
 process creation 1-12
 process data structures 1-10
 process execution 1-12
 process group
 send signal 2-413
 process group ID 2-373, 2-728, 2-731
 set 2-728
 process ID 2-373
 process identification 2-4
 process image
 new 2-226
 process priority

- automatic assignment 1-16
- change 2-508
- process statistics 2-22
- process suspension 2-542
- process termination 2-233
- process times
 - child 2-817
 - get 2-817
 - parent 2-817
- process-to-process communication 2-5
- process trace 2-567
- process user ID 2-731
- processor
 - difference, IBM Personal Computer AT and 032 Microprocessor C-7
- proc0 2-411
- proc1 2-411
- profil system call 2-560
- profile
 - execution time 2-560
- profile file 3-183
- profile setting 3-183
- profile, execution 2-456
- profiler, operating system 5-125
- program execution, DOS Services 2-167
- programmable character set font 3-88
- programmers workbench library 2-563
- programming with remote procedure calls 2-615
- protocol
 - definition 2-777
- protocol modes 5-79
- protocol procedure 2-869
- protoent structure 2-374
- pseudo-random number generator 2-585, 2-586
- pseudo-random numbers 2-206
- pseudo-terminal device 5-126
- ptrace system call 2-567
- pty special file 5-126
- publications
 - related xi
- push character back into input stream 2-841
- putc macro 2-572

- putchar macro 2-572
- putenv subroutine 2-575
- putgr subroutine 2-577
- putlong subroutine 2-654
- putp subroutine 2-136
- putpw subroutine 2-579
- puts subroutine 2-581
- putshort subroutine 2-654
- pututline subroutine 2-393
- putw subroutine 2-572
- putwc macro 2-572
- putwc subroutine 2-572
- putwchar macro 2-572
- putwchar subroutine 2-572
- putws subroutine 2-581

Q

- qconfig file 3-185
- qdaemon to backend interaction B-2
- qsort subroutine 2-583
- query DMA 5-67
- query hft device 2-814, 5-66
- query physical device 5-59
- query physical device identifiers 5-58
- query presentation space 5-65
- query terminal characteristics 2-814
- queue
 - message 2-703
 - send message to 2-490
- queue element
 - insert 2-406
 - remove 2-406
- queue identifier 2-468
- queue message
 - read 2-482
 - store 2-482
- queued message, issue 2-478
- queuing system B-1
- quick sort 2-583
- quit character 5-134

R

- rand subroutine 2-585
- random-number generator 2-585, 2-586
- random numbers 2-206
- random subroutine 2-586
- rasconf file 3-189
- raw subroutine 2-133, 2-270
- rcmd subroutine 2-588
- read
 - from a file, extended 2-591
 - message from a queue 2-482
 - next directory entry 2-146, 2-149
 - open a file to 2-538
- read a DOS file 2-186
- read a file tree 2-332
- read a password 2-370
- read attribute file stanza 2-79
- read from a file 2-591
- read input 2-597
- read lock 2-281
- read system call 2-591
- read/write file pointer
 - move 2-192, 2-434
- readdir subroutine 2-146, 2-149
- readv 2-597
- readx system call 2-591
- real user ID 2-729
- realloc subroutine 2-436
- reboot system call 2-599
- rebuild kernel 2-69
- receive
 - extended message from queue 2-493
- recv subroutine 2-601
- recvfrom subroutine 2-601
- recvmsg subroutine 2-601
- refresh subroutine 2-133, 2-270
- regcmp subroutine 2-604
- regex subroutine 2-604
- regex facility 2-610
- register-timer subroutine 2-670
- regular expression 2-604, 2-610
 - advance 2-610
 - compile 2-604, 2-610
 - match 2-604
 - step 2-610
- related publications xi
- relative path 1-28
- release
 - blocked signals 2-755
- relocation, a.out 3-9
- REM_control subroutine 2-665
- REM_process subroutine 2-667
- REM_quit subroutine 2-668
- REM_reset subroutine 2-668
- REM_start subroutine 2-668
- remainder function 2-290
- remote file access 2-10
- remote hft 5-95
- remote host
 - command execution 2-588, 2-659
- remote login 5-126
- remote procedure calls 2-615
 - authentication 2-617
 - communication model 2-616
 - terms 2-615
- Remote Procedure Calls (RPC)
 - asynchronous processing 2-635
 - authentication
 - authentication parameter 2-620
 - broadcasts 2-627
 - credentials 2-620, 2-623
 - credentials parameter 2-623
 - credentials, shorthand form 2-623
 - identifying caller 2-620
 - of server 2-621
 - opaque structure 2-623
 - permission checking 2-623
 - permissions 2-623
 - permissions, caller 2-620
 - permissions, refused 2-622
 - structure of 2-620
 - subroutines 2-625
 - verifier parameter 2-623
 - version number 2-620
 - version numbers 2-620
 - byte stream protocol 2-623
 - C programs 2-615
 - client
 - broadcast 2-627
 - matching replies 2-619

- permission checking 2-623
- recognition of messages 2-619
- RPC handle 2-628
- simulation programs 2-630
- data handling
 - deallocation 2-628
 - decoding 2-629
 - freeing 2-629
 - opaque data structures 2-620
 - record fragment 2-623
 - UDP/IP 2-632
- defined 2-615
- identifying remote programs 2-620
- Internet addresses 2-631
- IP 2-632
- message system
 - broadcasting 2-622, 2-627
 - call message 2-619
 - call, structure 2-619
 - discriminants 2-619, 2-621
 - error information 2-629
 - error messages, subroutines 2-629
 - error structure 2-629
 - overview 2-618
 - record marking 2-623
 - reply message 2-620
 - reply, error lists 2-621
 - reply, multiple 2-622
 - reply, rejected form 2-622
 - reply, simulating rejection 2-643
 - reply, structure of 2-620
 - reply, unnecessary 2-622
 - simulating RPC messages 2-641
 - transaction identifiers 2-619
 - union, discriminant 2-621
- overview 2-615
- ports
 - closing sockets 2-628
 - defined 2-624
 - internet address 2-631
 - mapping, destroying 2-634
 - mapping, removing 2-638
 - mapping, user interface 2-632
 - port number 2-632
 - reserved ports 2-624
 - socket structures 2-631
 - socket, opening 2-631
 - socket, pointer 2-631
 - socket, setting 2-631
- procedure number 2-620
- protocol compatibility 2-622
- protocol specification 2-620
- RPC message authentication 2-617
- RPC required XDR subroutines 2-641
 - xdr-accepted-reply 2-641, 2-642
 - xdr-callhdr 2-642
 - xdr-callmsg 2-642
 - xdr-opaque-auth 2-643
 - xdr-pmap 2-643
 - xdr-pmaplist 2-643
 - xdr-rejected-reply 2-643
 - xdr-replymsg 2-644
- RPC subroutines 2-624-2-641
 - auth-destroy 2-625
 - authnone-create 2-625
 - authunix-create 2-625
 - authunix-create-default 2-626
 - broadcasting 2-627
 - callrpc 2-626
 - clnt-broadcast 2-627
 - clnt-call 2-628
 - clnt-destroy 2-628
 - clnt-freeres 2-629
 - clnt-geterr 2-629
 - clnt-pcreateerror 2-629
 - clnt-perrno 2-630
 - clnt-perror 2-630
 - clntraw-create 2-630
 - clnttcp-create 2-631
 - clntudp-create 2-631
 - common parameters 2-624
 - error routines 2-629
 - get-myaddress 2-632
 - overview 2-624
 - pmap-rmtcall 2-633
 - pmap-set 2-634
 - pmap-unset 2-634
 - registerrpc 2-634
 - rpc-createerr 2-635
 - simulation routine 2-630
 - svc-destroy 2-635
 - svc-fds 2-635

- svc-freeargs subroutine 2-635
- svc-getargs 2-636
- svc-getcaller 2-636
- svc-register 2-637
- svc-run 2-637
- svc-sendreply 2-637
- svc-unregister 2-638
- svcerr-auth 2-638
- svcerr-decode 2-638
- svcerr-noproc 2-639
- svcerr-noprog 2-639
- svcerr-progvers 2-639
- svcerr-systemerr 2-639
- svcerr-weakauth 2-640
- svcrawl-create 2-640
- svctcp-create 2-640
- svcudp-create 2-641
- xprt-register 2-644
- xprt-unregister 2-644
- server
 - broadcast 2-627
 - internet address 2-631
 - permission checking 2-623
 - RPC handle 2-628
 - simultaneous request servicing 2-620
- simulating remote programs 2-630
- remote requests
 - authentication 2-588
- remove
 - directory entry 2-843
 - queue element 2-406
- remove a DOS Services directory 2-190
- remove protocol procedure 2-869
- remque subroutine 2-406
- rename a DOS file 2-188
- rename system call 2-649
- reopen all files 2-200
- replace mode 5-86
- report CPU time used 2-100
- reposition the file pointer of a stream 2-324
- res-init subroutine 2-654
- res-mkquery subroutine 2-654
- res-send subroutine 2-654
- reset
 - directory pointer 2-146, 2-149
- resetterm subroutine 2-133

- resetty subroutine 2-133, 2-270
- resolution
 - path name 1-26
- resolver 2-655
- resolver subroutines 2-654
- retrieve a message, insert, or help text 2-486
- return login name of user 2-431
- return node ID 2-209
- return node nickname 2-209
- return status 1-33
- rewind subroutine 2-324
- rewinddir subroutine 2-146, 2-149
- rewrite existing file 2-110
- rexec subroutine 2-659
- rindex subroutine 2-401
- Ring Error Monitor subroutine library 2-661
- ring, screen manager 5-70
- rmdir system call 2-692
- root directory
 - change 2-97
- routine libraries
 - See libraries
- routines
 - See kernel subroutines
 - See system calls and subroutines
- RPC protocol 2-617
 - assigning procedure numbers 2-618
 - assigning program numbers 2-618
 - assigning version numbers 2-618
- IP
 - pmap-getport 2-632
- rresvport subroutine 2-588
- RT hardware access C-7
- ruserok subroutine 2-588

S

- saveterm subroutine 2-133
- savetty subroutine 2-133, 2-270
- sbrk system call 2-48
- scan
 - directory 2-701
- scan directory pointer 2-86
- scandir subroutine 2-701

scanf subroutine 2-694
 scanw subroutine 2-133, 2-271
 schedule alarm 2-25
 screen handling package 2-129
 screen manager ring 5-70
 screen optimization package 2-129
 scroll subroutine 2-133, 2-271
 scrollok subroutine 2-133, 2-271
 search and update, linear 2-432
 search trees, binary 2-829
 search, binary 2-50
 security description file 3-207
 security file 3-35
 seed48 subroutine 2-206
 seekdir subroutine 2-146, 2-149
 segment
 data 1-8
 stack 1-9
 text 1-8
 sel_attr subroutine 2-271
 select support 5-27, 5-45, 5-127, 5-144
 select system call 2-703
 semaphores 2-707, 2-711, 2-714
 semctl system call 2-707
 semget system call 2-711
 semop system call 2-714
 send
 message to message queue 2-490
 signal to a process 2-411
 signal to process group 2-411
 send a message to a queue 2-490
 send subroutine 2-718
 sendmail
 configuration file 3-193
 sendmail.cf file format 3-193
 sendmsg subroutine 2-718
 sendto subroutine 2-718
 servent structure 2-381
 server
 See Remote Procedure Calls (RPC)
 serverattach attributes 3-204
 serverattach file format 3-203
 set-group-ID mode bit 2-228
 set time 2-794
 set-user-ID mode bit 2-228
 set_term subroutine 2-133
 setbuf subroutine 2-720
 setbuffer subroutine 2-722
 seteuid subroutine 2-729
 setgid system call 2-731
 setgrent subroutine 2-349
 setgroups system call 2-724
 sethostent subroutine 2-353
 sethostid socket subroutine 2-356
 sethostname socket subroutine 2-358
 setjmp subroutine 2-726
 setlinebuf subroutine 2-722
 setlogmask subroutine 2-807
 setnetent subroutine 2-364
 setpgrp system call 2-728
 setprotoent subroutine 2-374
 setpwent subroutine 2-376
 setreuid subroutine 2-729
 setruid subroutine 2-729
 setscrreg subroutine 2-134
 setservent subroutine 2-381
 setsockopt socket subroutine 2-385
 setstate subroutine 2-586
 setterm subroutine 2-134, 2-272
 setting environment 3-183
 setting the profile 3-183
 setuid system call 2-731
 setup_attr subroutine 2-272
 setupterm subroutine 2-136
 setutent subroutine 2-393
 setvbuf subroutine 2-720
 sgetl subroutine 2-733
 shared memory
 control operations 2-738
 shared memory segment
 attach 2-734
 detach 2-741
 get 2-743
 sharing files 2-10
 shell command, issue 2-810
 shell environment 2-227
 shell variable 2-227
 shell variable, value of 2-347
 shift, single 4-9
 shmat system call 2-734
 shmctl system call 2-738
 shmdt system call 2-741

shmget system call 2-743
 shmop system calls 2-734, 2-738, 2-741, 2-743
 shorten a file 2-330
 shut down
 socket connection 2-746
 shutdown socket subroutine 2-746
 SIGAIO signal 2-751
 SIGALRM signal 2-751
 sigblock system call 2-748
 SIGBUS signal 2-750
 SIGCLD signal 2-233, 2-234, 2-481, 2-751, 2-753, 2-873
 SIGDANGER signal 2-750, 2-753
 SIGFPE signal 2-314, 2-750, 2-753, 2-762
 SIGGRANT signal 2-751
 SIGHUP signal 2-234, 2-750
 SIGILL signal 2-750
 SIGINT signal 2-750
 SIGIOINT signal 2-751
 SIGIOT signal 2-750
 SIGKILL signal 2-230, 2-748, 2-750
 SIGMSG signal 2-751
 signal
 block delivery 2-748
 process group 2-413
 signal action 2-750
 signal-catching function 2-750
 signal facilities
 enhanced 2-761
 signal handler 2-750
 signal mask
 setting 2-757
 signal overview 2-5
 signal stack context 2-759
 signal system call 2-750
 signals 1-16, 2-750, 2-755
 release blocked 2-755
 signals, floating-point 2-313
 signals, software 2-781
 sigpause system call 2-755
 SIGPIPE signal 2-750
 SIGPTY signal 2-751
 SIGPWR signal 2-751, 2-753
 SIGQUIT signal 2-750
 SIGRETRACT signal 2-751
 SIGSEGV signal 2-750
 sigsetmask system call 2-757
 SIGSOUND signal 2-751
 sigstack system call 2-759
 SIGSYS signal 2-750
 SIGTERM signal 2-751
 SIGTRAP signal 2-750
 SIGUSR1 signal 2-751
 SIGUSR2 signal 2-751
 sigvec system call 2-761
 sin subroutine 2-764
 sine function 2-764
 single-byte controls 4-11
 single-shift control 4-9
 single-user mode 3-3
 sinh subroutine 2-766
 sleep subroutine 2-767
 slocal program 3-151
 sockaddr structure 2-778
 socket
 bind to privileged address 2-588
 create 2-769
 definition 2-776
 initiate a connection 2-103
 socket connection
 accept 2-17
 listen 2-420
 shut down 2-746
 socket message
 receive 2-601
 send 2-718
 socket name 2-383
 bind 2-46
 socket options 2-385
 socket peer name 2-371
 socket subroutine 2-769
 socketpair subroutine 2-772
 sockets
 overview 2-776
 routines 2-774
 sockets subroutine library 2-774
 software
 enhanced signal facilities 2-761
 software signals 2-781
 sort, array 2-26
 sort, quick 2-583
 sound data 5-83

space
 allocation change for data segment 2-48
 space subroutine 2-549
 special character definitions for eqn and neqn 4-56
 special file 2-460
 create 2-450
 special files 1-20, 1-36
 specification of text file format 3-108
 sprintf subroutine 2-553
 sputl subroutine 2-733
 sqrt subroutine 2-235
 square root 2-235
 srand subroutine 2-585
 srandom subroutine 2-586
 srand48 subroutine 2-206
 sscanf subroutine 2-694
 sscanf subroutine 2-781
 SS1-SS4 4-9
 stack
 signal 2-759
 stack segment 1-9
 standard I/O 2-572
 standard I/O subroutine library 2-783
 standard interprocess communication package 2-328
 standend subroutine 2-134, 2-272
 standout subroutine 2-134, 2-272
 stanza, add 2-71
 stanza, delete 2-75
 stanza, read 2-79
 stanza, replace 2-71
 stanza, write 2-71
 start
 character 5-134
 system 2-599
 virtual machine 2-409
 stat structure 4-73
 stat system call 2-786
 stat.h header file 4-73
 state, code service 3-203
 statfs system call 2-790
 statistics
 file system 2-848
 process 2-22
 statistics, file system 2-848

status
 check I/O 2-703
 file 2-786
 file system 2-790
 symbolic link 2-786
 status of a DOS file, get 2-194
 status of DOS Services device, get 2-202
 status, stream 2-285
 statusfile parameter B-2
 stdio subroutine library 2-783
 stdipc (ftok subroutine) 2-328
 stime system call 2-794
 stop
 wait for child prcoess to 2-874
 wait for child process to 2-872
 stop character 5-134
 storage image file 3-43
 store
 message from a queue 2-482
 store subroutine 2-141
 strcat subroutine 2-795
 strchr subroutine 2-795
 strcmp subroutine 2-795
 strcpy subroutine 2-795
 strcspn subroutine 2-795
 stream closing and flushing 2-278
 stream I/O 2-572
 stream open 2-291
 stream status 2-285
 stream, assigning buffering to 2-720
 stream, data
 3270 5-26
 stream, get character or word from 2-342
 string from a stream, obtaining 2-379
 string handling 2-516
 string operations 2-52, 2-401, 2-526, 2-795
 international character support 2-503
 string to integer conversion 2-800
 string, write to a stream 2-581
 strlen subroutine 2-795
 strncat subroutine 2-795
 strncmp subroutine 2-795
 strncpy subroutine 2-795
 strpbrk subroutine 2-795
 strrchr subroutine 2-795
 strspn subroutine 2-795

- strstr subroutine 2-795
- strtod subroutine 2-798
- strtok subroutine 2-795
- strtol subroutine 2-800
- structures
 - _res 2-655
 - a.out 3-5
 - a.out relocation 3-9
 - accounting file 3-15
 - archive file member 3-18
 - backup 3-24
 - cpio 3-45
 - Define_Code SVC 5-22
 - device driver config 5-23
 - devinfo 3-66, 5-119
 - directory file 3-69
 - disk config 5-22
 - diskette customizing 5-33
 - fd devinfo 5-34
 - fullstat 4-60
 - HD devinfo 5-37
 - hostent 2-354
 - inode 3-119
 - ipc_perm 2-7
 - lprio.h 5-119
 - lprmode 5-119
 - LPRUDE 5-120
 - minidisk customize 5-36
 - mnttab.h 3-161
 - msghdr 2-602
 - netent 2-364
 - ob_text 2-677
 - oprmode 5-120
 - output_buf 2-677
 - portstatus 3-178
 - process data 1-10
 - protoent 2-374
 - servent 2-381
 - sockaddr 2-778
 - stat 4-73
 - superblock 3-101
 - symbol table 3-10
 - tacct.h 3-16
 - tape archive header 3-217
 - termio 5-135
 - VRM Query-Device call 5-23

- structures, file
 - See file formats
- subroutine libraries
 - See libraries
- subroutines
 - See also kernel subroutines
 - del_ipc_prof 2-143
 - find_ipc_prof 2-287
- subroutines, RPC
 - See Remote Procedure Calls (RPC)
- subroutines, XDR
 - See eXternal Data Representation (XDR)
- subsystem
 - buffer 1-32
 - file i/o 1-32
- subwin subroutine 2-134, 2-272
- superblock 1-21
 - update 2-806
- superbox subroutine 2-273
- supervisor calls, AIX
 - See system calls and subroutines
- suspend
 - process 2-542
- suspend execution 2-767
- SVCs, AIX
 - See system calls and subroutines
- swab subroutine 2-802
- swap bytes 2-802
- symbol table structure 3-10
- symbolic link system call 2-803
 - symbolic link 2-803
- symbols, display 4-26
- sync system call 2-806
- synchronize a DOS file 2-171
- syntax ix
- sys_errlist 2-543
- sys_nerr 2-543
- sysck.cfg file 3-207
- syslog subroutine 2-807
- system
 - reboot 2-599
- system calls
 - difference from subroutines 2-2
 - errno values A-1
 - functional summary 2-2
- system calls and subroutines

- system data types 4-78
- system error messages 2-543
- system file 3-210
- system log
 - make entry 2-807
- system name
 - extended 2-839
 - get 2-839
- system profiler 5-125
- system subroutine 2-810
- system volume format 3-100

T

- TABDLY 5-137
- table
 - call switch 1-32
 - device switch 1-32
 - DOS function call C-2
- table, mounted file system 3-161
- tabs, non-standard 3-108
- TAB0 5-137
- TAB1 5-137
- TAB2 5-137
- TAB3 5-137
- tacct.h structure 3-16
- tail, of screen manager ring 5-70
- tan subroutine 2-764
- tangent function 2-764
- tanh subroutine 2-766
- tape archive header structure 3-217
- tape driver file 5-130
- tape special file 5-130
- tar file 3-217
- tcflsh 5-146
- tcgeta 5-144
- TCP/IP
 - See Remote Procedure Calls (RPC)
- tcsbrk 5-146
- tcseta 5-145
- tcsetaf 5-145
- tcsetaw 5-145
- tcxonc 5-146
- tdelete subroutine 2-829

- tellmdir subroutine 2-146, 2-149
- tempnam subroutine 2-820
- temporary file creation 2-819
- temporary file creation, DOS 2-180
- temporary file naming 2-820
- TERM environment variable 4-48, 4-76
- TERM variable 2-815
- termcap
 - emulation using terminfo 2-137
- termdef subroutine 2-814
- terminal capability data base 3-219
- terminal characteristics 2-814
- terminal file name generation 2-118
- terminal interface control 5-153
- terminal name 2-832
- terminal, data base 3-219
- terminate
 - wait for child process to 2-872, 2-874
- terminate a process 2-233
- terminfo file 3-219
- termio file 5-133
- termio structures 5-135
- termio support 5-44
- text
 - lock 2-547
 - unlock 2-547
- text file format specification 3-108
- text segment 1-8
- text, help, issue 2-471
- tgetent subroutine 2-137
- tgetflag subroutine 2-137
- tgetnum subroutine 2-137
- tgetstr subroutine 2-137
- tgoto subroutine 2-137
- thresholds, locator 5-81
- tilde accent character 4-10
- time
 - get 2-816
 - obtain 2-389
 - set 2-794
- time format 2-530
- time profile
 - execution time 2-560
- time structure 2-533
- time system call 2-816
- time to string conversion 2-120

time used report, CPU 2-100
 time_t data type 4-78
 times system call 2-817
 timezone external variable 2-120
 tmpfile subroutine 2-819
 tmpnam subroutine 2-820
 toascii subroutine 2-105
 tojhira subroutine 2-105
 tojkata subroutine 2-105
 tojlower 2-105
 _tojlower subroutine 2-105
 tojupper 2-105
 _tojupper subroutine 2-105
 token ring
 See Ring Error Monitor subroutine library
 tolower subroutine 2-105
 _tolower subroutine 2-105
 touchwin subroutine 2-134, 2-273
 toupper subroutine 2-105
 _toupper subroutine 2-105
 tparm subroutine 2-136
 tputs subroutine 2-136, 2-137
 trace 2-822
 execution of a process 2-567
 trace channel, check whether enabled 2-822
 trace collector, AIX 2-827
 trace daemon 2-824
 trace driver 5-150
 trace special file 5-150
 trace_on subroutine 2-822
 traced process
 control 2-567
 traceoff subroutine 2-134
 traceon subroutine 2-134
 trackloc subroutine 2-274
 trailer record 3-46
 translate
 characters 2-105, 2-514
 group IDs 2-93
 owner IDs 2-93
 trap, floating-point exception 2-313
 trap, kernel 1-31
 trc_start subroutine 2-824
 trc_stop subroutine 2-824
 trcunix subroutine 2-827
 trdconf 3-242

trdfile 3-244
 tree, read 2-332
 trees, binary search 2-829
 trigonometric functions 2-764
 tsearch subroutine 2-829
 tstp subroutine 2-274
 tty special file 5-153
 ttyname subroutine 2-832
 twalk subroutine 2-829
 typeahead subroutine 2-134
 types.h header file 4-78
 TZ environment variable 4-48
 tzname external variable 2-120
 tzset subroutine 2-120

U

U.S. English keyboard 5-98
 uint data type 4-78
 ulimit system call 2-834
 ulong data type 4-78
 umask system call 2-836
 umlaut accent character 4-10
 umount system call 2-837
 uname system call 2-839
 unamex system call 2-839
 unctrl subroutine 2-134, 2-274
 ungetc subroutine 2-841
 ungetwc subroutine 2-841
 Unix error collector 2-224
 unlink a DOS file 2-198
 unlink system call 2-843
 unlock
 region of a file 2-427
 unlock a region of a DOS file 2-176
 unmount
 file system 2-837
 unregister-timer subroutine 2-670
 update
 delayed blocks 2-806
 i-nodes 2-806
 superblock 2-806
 update, linear 2-432
 user ID

- get 2-391
- set 2-731
- set effective 2-729
- set real 2-729
- user information 2-846
- user information name, find value 2-392
- user limits 2-834
- user login name 2-140
- user login name, obtaining 2-431
- user mode 1-6
- user mode addressing 1-8
- user name 2-140
- ushort data type 4-78
- usrinfo system call 2-846
- ustat system call 2-848
- utime system call 2-850
- utimes subroutine 2-853
- utmp file 3-246
- utmp file entry access 2-393
- utmp file, find user's slot 2-833
- utmpname subroutine 2-393
- uvmount system call 2-855

V

- value of environment variable 2-347
- value of user information name, find 2-392
- values.h header file 4-80
- varargs argument list, print 2-866
- varargs macro 2-858
- variable-length parameter list 2-858, 2-866
- verify program assertion 2-28
- verify, write 2-443
- vfprintf subroutine 2-866
- vfs 3-249
- vidattr subroutine 2-137
- vidputs subroutine 2-136
- virtual
 - memory 1-6
 - terminal data (VTD) 5-78
- virtual file system 3-249
- virtual machine
 - IPL 2-409
 - restart 2-599

- start 2-409
- wait for termination 2-409
- virtual memory image 5-122
- vprintf subroutine 2-866
- vrcppr subroutine 2-869
- VRM device driver 1-32
- VRM Query_Device call 5-23
- vscroll subroutine 2-274
- vsprintf subroutine 2-866
- VTD 5-78
- VTDL 5-137
- VT0 5-137
- VT1 5-137

W

- waddch subroutine 2-130, 2-242
- waddfld subroutine 2-243
- waddstr subroutine 2-130, 2-243
- wait
 - for I/O activity 2-703
 - for signal 2-542
- virtual machine 2-409
- wait system call 2-872
- waitvm system call 2-409
- wait3 subroutine 2-874
- walk a file tree 2-332
- watof subroutine 2-883
- watoi subroutine 2-885
- watol subroutine 2-885
- wattroff subroutine 2-134
- wattron subroutine 2-134
- wattrset subroutine 2-134
- wchgat subroutine 2-244
- wclear subroutine 2-134, 2-245
- wclrtoebot subroutine 2-134, 2-246
- wclrtoeol subroutine 2-134, 2-246
- wcolorend subroutine 2-246
- wcolorout subroutine 2-247
- wdelch subroutine 2-134, 2-248
- wdeleteln subroutine 2-134, 2-248
- well-known port 2-589
- werase subroutine 2-134, 2-262
- wgetch subroutine 2-134, 2-263

wgetstr subroutine 2-134, 2-264
 winch subroutine 2-134, 2-264
 winsch subroutine 2-134, 2-265
 winsertln subroutine 2-134, 2-265
 wmove subroutine 2-134, 2-267
 wnoutrefresh subroutine 2-134
 word, get from stream 2-342
 workbench library 2-563
 wprintw subroutine 2-134, 2-270
 wrefresh subroutine 2-134, 2-270
 write
 file to permanent storage 2-326
 open a file to 2-538
 to a file 2-876, 2-880
 write a string to a stream 2-581
 write characters 2-572
 write group file entry 2-577
 write lock 2-281
 write password file entry 2-579
 write stanza 2-71
 write system call 2-876
 write to a DOS file 2-204
 write to a stream 2-572
 write-verify 2-443
 write words 2-572
 writev 2-880
 writex system call 2-876
 wscanw subroutine 2-135, 2-271
 wsetscrreg subroutine 2-135
 wstandend subroutine 2-135, 2-272
 wstandout subroutine 2-135, 2-272
 wstrtod subroutine 2-883
 wstrtol subroutine 2-885
 wtmp file 3-246

X

XCASE 5-140

Y

yes/no response, determine forms 2-536
 YP_MAP_X_LATE 3-251
 translated YP map names 3-251
 y0, y1, yn subroutines 2-44

Z

zombie process 2-233

Numerics

2-byte characters 4-9
 3-byte integer conversion to long
 integers 2-416
 3270 data stream 5-26
 3270 device driver 5-26
 3270 Distributed Function Terminal 5-26
 3278/79 Emulation Adapter 5-26
 4.3BSD
 68881 floating-point processor 2-295, 2-308,
 2-315

IBM RT AIX Operating System Technical Reference

Book Title

SC23-2126-0

Order No.

Book Evaluation Form

Your comments can help us produce better books. You may use this form to communicate your comments about this book, its organization, or subject matter, with the understanding that IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you. Please take a few minutes to evaluate this book as soon as you become familiar with it. Circle Y (Yes) or N (No) for each question that applies and give us any information that may improve this book.

Y N Is the purpose of this book clear?

Y N Is the table of contents helpful?

Y N Is the index complete?

Y N Are the chapter titles and other headings meaningful?

Y N Is the information organized appropriately?

Y N Is the information accurate?

Y N Is the information complete?

Y N Is only necessary information included?

Y N Does the book refer you to the appropriate places for more information?

Y N Are terms defined clearly?

Y N Are terms used consistently?

Y N Are the abbreviations and acronyms understandable?

Y N Are the examples clear?

Y N Are examples provided where they are needed?

Y N Are the illustrations clear?

Y N Is the format of the book (shape, size, color) effective?

Other Comments

What could we do to make this book or the entire set of books for this system easier to use?

Optional Information

Your name

Company name

Street address

City, State, ZIP

No postage necessary if mailed in the U.S.A.

Tape

Please Do Not Staple

Tape

Cut or Fold Along Line

Fold and Tape

Fold and Tape

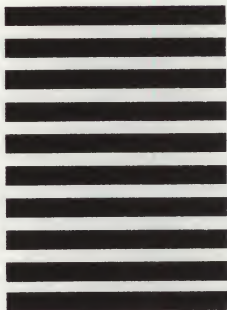
International Business Machines Corporation
Department 997, Building 998
11400 Burnet Rd.
Austin, Texas 78758-3493

POSTAGE WILL BE PAID BY ADDRESSEE

FIRST CLASS PERMIT NO. 40 ARMONK, NEW YORK

BUSINESS REPLY MAIL

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES





The IBM AIX/RT Family.

Reader's Comment Form

IBM RT AIX Operating System Technical Reference

SC23-2126-0

Your comments assist us in improving our products. IBM may use and distribute any of the information you supply in any way it believes appropriate without incurring any obligation whatever. You may, of course, continue to use the information you supply.

For prompt resolution to questions regarding set up, operation, program support, and new program literature, contact your IBM representative, your IBM authorized dealer, or your IBM authorized remarketer.

Comments:

Tape

Please Do Not Staple

Tape

Cut or Fold Along Line

Fold and Tape

Fold and Tape

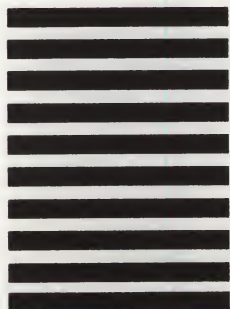
International Business Machines Corporation
Department 997, Building 998
11400 Burnet Rd.
Austin, Texas 78758-3493

POSTAGE WILL BE PAID BY ADDRESSEE

FIRST CLASS PERMIT NO. 40 ARMONK, NEW YORK

BUSINESS REPLY MAIL

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES





© IBM Corp. 1988
All rights reserved.

International Business
Machines Corporation
11400 Burnet Road
Austin, Texas 78758

Printed in the
United States of America

SC23-2126-0

SC23-2126-0



27F4355

IBM
®